IEEE MICRO

Chips, Systems, Software, and Applications

OCTOBER 1993

A WORLD OF EMBEDDED SYSTEMS:

3 Powerful New Chips From Japan

SPECIAL FEATURES:

- MOTOROLA POWER PC
- END-USER PERFORMANCE, PART 2
- DISTRIBUTED SYSTEM PERFORMANCE

IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IEEE MICRO

Published by the IEEE Computer Society

Volume 13 Number 5 October 1993

FEATURES

SPECIAL FEATURES

Cover Image: Francesco Ruggeri, Image Bank West

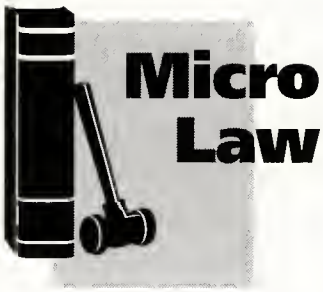
Cover Design: Design and Direction

Circulation: *IEEE Micro* (ISSN 0272-1732) is published bimonthly by the IEEE Computer Society, PO Box 3014, Los Alamitos, CA 90720-1264; IEEE Computer Society Headquarters, 1730 Massachusetts Ave., NW, Washington, DC 20036-1903; IEEE Headquarters, 345 East 47th St., New York, NY 10017. Annual subscription: $23 in addition to IEEE Computer Society or any other IEEE society member dues; $42 for members of other technical organizations. This journal is also available in microfiche form.
Postmaster: Send address changes and undelivered copies to *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. Second-class postage is paid at New York, NY, and at additional mailing offices. Canadian GST#125634188.

DEPARTMENTS

IEEE Computer Society
PO Box 3014, Los Alamitos, CA 90720-1264
(714) 821-8380; menglish@computer.org

Editorial: Unless otherwise stated, bylined articles and descriptions of products and services reflect the author's or firm's opinion; inclusion in this publication does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society. Send editorial correspondence to *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. All submissions are subject to editing for style, clarity, and space considerations.



Protecting industrial property rights

Richard H. Stern

Graham & James

2000 M Street, N.W.

Suite 700, Washington,

DC 20036-3307

> **modeling.** Technique of system analysis and design using mathematical or physical idealizations of all or a portion of the system. Completeness and reality of the model are dependent on the questions to be answered, the state of knowledge of the system, and its environment.
>
> **model.** A mathematical or physical representation of system relationships. *See*: mathematical model.
>
> **mathematical model (analog computers).** A set of equations used to represent a physical system.
>
> —*IEEE Standard Dictionary of Electrical and Electronics Terms* (3rd ed., 1984)

Legal systems can be represented in terms of models, too. The models are not sets of equations, however. They are a set of descriptions of the system or its characteristics. Thus, the US patent and copyright law systems can be described, as they are here, in terms of models. Representing legal systems in terms of models facilitates understanding of their operation and permits some amount of simulation. That may permit us to ascertain the circumstances that impose stresses on the systems and may strain them beyond their limits. The results of simulations may also suggest modifications of the modeled system, to avoid or lessen the effects of stressful circumstances.

The conventional approach to intellectual property protection in the United States has relied almost entirely on two models—those of patent law and copyright law. These two models of legal protection have limitations, however, particularly in regard to late 20th century computer software technology. Systems based on these models are particularly unsuited to protecting noncode aspects of computer software—to protecting against nonverbatim, nonliteral copying of computer programs. The time may now be ripe for consideration of other legal models for protecting software, at least nonliteral, noncode aspects of software.

The patent model

The patent law model for industrial property rights requires as conditions of legal protection: utility, novelty, and a high level of technical merit or technological advance. That high level of advance is variously called inventive level, inventive step, or nonobviousness. It refers to work substantially above the routine, an advance in technology that a person of ordinary skill in the field would not have conceived and reduced to working form.

To determine whether a product that is a candidate for patent protection meets the conditions for protection, a patent system relies on examination by technical experts before any legal protection becomes effective. This is an expensive and time-consuming procedure, both for the applicant and the government. But the filtration procedure is considered worth the high front-end costs, for several reasons.

One reason is that it relieves the courts of the burden of making such an assessment in the first instance, a task for which they are not well equipped. Another reason is that competitors, potential investors, and the general public have certainty, or at least substantial assurance, that an issued patent covers a true invention and thus a valid, enforceable piece of intellectual property.

The certainty of a patent is further assured by patent law's requirement that a patent applicant

0272-1732/93/1000-002$03.00 © 1993 IEEE

must define the subject matter to be protected, by use of claims describing the scope of the patent right. The claims of a patent provide a description of that which others must not make, use, or sell. By the same token, a patent's claims tell the public what is *not* staked out as the exclusive right of the patent's owner, and is therefore available to competitors.

Patent law does not protect ideas, as such, and in principle the system of protection is limited to machines and other products implementing or carrying out novel ideas. Thus, laws of nature and mathematical principles cannot be monopolized under a patent. By the same token, algorithms and computer programs, as such, cannot be patented. (A vast amount of ingenuity has been devoted, however, to obtaining patents on algorithms and computer programs; for example, by claiming them as "machine systems" or other euphemisms. This has resulted in considerable litigation.)

Other characteristics of the patent model include the right, for approximately 20 years, to exclude others from making, using, importing, and selling the patented subject matter. This patent property right against others is absolute, in the sense that independent invention is not a defense to a claim for patent infringement. (If you patent a widget, and later I independently design the same widget and start manufacturing and selling it, I am an infringer and you can shut me down.)

Finally, and most important for present purposes, a patent property right against others is additionally absolute in the sense that a patentee is free to withhold the use of the invention from others, entirely or selectively. Moreover, a patentee is ordinarily entitled not only to damages for patent infringement but also an injunction to prevent any future unconsented-to use of the invention. Subject to very narrow qualification, the US model of patent protection (unlike that of other countries) does not permit compulsory licensing or anything like it.

The copyright model

The copyright law model of legal protection requires only a very minimal level of merit or creativity—little more than failure wholly to plagiarize the work from another author. Accordingly, there is no need for examination and filtering by technical experts before legal protection attaches to a work. Ordinarily, the first time that the creative level of a work is examined is in the course of a copyright infringement suit before a federal court. The result is lower front-end cost for copyright protection, but possibly much higher costs in litigation, if that ultimately occurs.

A copyright owner has the right, typically for a 75-year term, to exclude others from reproducing, importing, and distributing the copyrighted work. Apart from control over public performance and display, however, copyright law does not prevent use of the subject matter. Thus, use of infringing software (for example, execution of an illegally copied program) is not copyright infringement unless the use involves the making of a copy. (One major court has held, however, that loading a computer program into RAM is the making of a fixed copy and thus an act of infringement. See Micro Law, *IEEE Micro*, June 1993.)

Independent creation of the subject matter is a complete defense, however, unlike independent invention for patents. If you write some code identical to my copyrighted code, without ever having seen my code, you are not liable to me as a copyright infringer.

Like a patent, a copyright confers an absolute property right, in the sense that a copyright owner may at will withhold from others, entirely or selectively, the right to reproduce the work. A copyright owner may secure both damages from and injunctions against those who infringe the copyright.

US law does not ordinarily permit, and the Berne Convention (a treaty to which the US recently became a signatory) prohibits, compulsory licensing of copyrighted works. The Berne Convention also prohibits discrimination against some kinds of literary work (as which computer programs are classified) by giving them less favored treatment than other kinds of literary work.

A copyright, unlike a patent, has no claims defining its scope. Accordingly, the scope of a copyright is whatever a court ultimately holds that it is. That scope is largely unpredictable in advance. Ordinarily, a copyright will protect against verbatim copying, such as bit-for-bit or instruction-for-instruction copying of code, but that is often not the major issue in software cases. In recent years, ingenious counsel have frequently persuaded courts to protect against nonliteral, nonverbatim imitation of copyrighted works—for example, imitation of user interfaces and command languages for application programs. This has been done on the imaginative theory that computer programs and other works of new technology deserve broad protection analogous to that accorded poems and novels, for which copyright law may protect plot and other nonliteral aspects.

These developments have created great tension with the basic legal principle that copyright protects only particular expressions of ideas. Copyright does not protect any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in a work. (This list is what we may call the *index prohibitorum* of section 102(b) of the US Copyright Act. That section expressly states that the scope of copyright does not extend to the foregoing list of things.)

In allowing such "nonliteral" protection of software, some courts have confused copyrights with patents. (The *Whelan* and *Lotus-Paperback* decisions are examples of this.) Very recent decisions have tended in the opposite direction, however, and have restored the previously recognized differences between the copyright and patent

continued on p. 100

Programming Windows

Richard Mateosian

2919 Forest Avenue

Berkeley, CA

94705-1310

(510) 540-7745

Windows has recently grown tremendously in popularity. One large obstacle stands in the way of continued growth, and Microsoft has mounted full-scale assaults on that obstacle from several directions.

The obstacle is the steep learning curve that application programmers face when they want to work in the Windows environment. You can't write an application that displays "Hello, world" in a window without encountering arcane concepts. The new application programmer must confront the intricacies of object orientation and C++, over a thousand system calls, and the huge foundation class library.

The serious programmmer's guide to this mess has been and continues to be Charles Petzold's *Programming Windows*, now in its third edition (Microsoft, 1992). This book has received universal praise, so I don't need to add anything here. If you want to program Windows applications and haven't read this book, you need to do so. Nothing in this column will change that.

Another book you should read if you want to program Windows applications is *The Windows Interface—An Application Design Guide* (Microsoft, 1992). In it, Microsoft promotes visual and functional consistency across Windows applications. Many developers hope to make their products "intuitive" to users to reduce users' resistance to learning them. These developers often forget that most of what makes a user interface intuitive is its similarity to other interfaces that the user already knows. Adhering to the standards in this book can go a long way toward making your interface easy to learn.

Visual C++, Professional Edition (Microsoft, Redmond, Wash.; $499)

Visual C++ is a package that contains everything you can get from Microsoft to program Windows applications. You can get it on CD, but the usual distribution medium is high-density diskettes—20 of them, filled with compressed files. It expands to 56 Mbytes on your hard disk. The accompanying 11 books weigh around 40 pounds.

If you're a Windows programmer with experience using C and the Windows Software Development Kit (SDK), you don't have to give them up. They're included. If you want to develop DOS applications or generate p-code or use the CodeView debugger for DOS or Windows, don't worry. It's all included.

While not abandoning the SDK, Microsoft has provided an alternate paradigm that should narrow its use. Visual C++ automates the creation of skeleton Windows applications based on the Microsoft Foundation Class Library, Version 2.0. You still need to call SDK functions as you hang flesh on the skeleton.

You invoke the Class Wizard and specify a few options in a simple dialog box, and then Visual C++ gives you a skeleton application containing a great deal of functionality. You can immediately resize or scroll your new application's window and create or save documents using your application's File menu or toolbar. You can then use the App Studio to create or modify resources like icons, menus, and toolbars. App Studio lets you manipulate these resources directly, so you rarely need to work with the resulting source files.

Your new skeleton application uses the C++ language and the Microsoft foundation classes. You can accomplish essentially the same things using C and the SDK, but C++ and the foundation classes provide a cleaner approach. For example, the constructor and destructor functions of foundation classes automatically contain calls

0272-1732/93/1000-0004$03.00 © 1993 IEEE

that you must remember to make explicitly with C and the SDK.

Visual C++ is an integrated environment with debugger, source browser, and text editor. The embedded editor uses color to emphasize language elements, and it highlights error lines after failed compilations. The downside, however, is that you can't replace the embedded editor with your own. Many programmers will suffer serious withdrawal symptoms if they can't use Brief, the outstanding text editor from Solution Systems, or Emacs, the popular Gnu editor. No doubt clever programmers will develop ways to let Brief and Emacs in the back door, and Microsoft will reluctantly incorporate those options into a future release.

Recognizing how new and how hard Visual C++ will seem to most programmers, Microsoft has bent over backwards to make it easy to use. Visual C++ has extensive on-line help. It is well organized, well written, and makes good use of automated cross-referencing (hypertext). Visual C++ also contains an excellent tutorial in which you can build a drawing program from scratch. A collection of progressively more advanced project directories allows you to check your progress in this tutorial or simply follow along without typing anything.

Visual C++ includes a 36-page "magazine," which introduces its features through interviews with members of its project team. This is an extremely helpful overview, and you should take the time to read it.

There is another way in which Visual C++ is easy to use. Microsoft C development systems have been notoriously hard to install and get running. When I reviewed the Borland C++ package (August 1992), I wasn't able to include a review of the Microsoft C development system, because I've never been able to get it to compile even the sample programs that came with it. I didn't have that problem with Visual C++. It didn't work immediately, but I was able to get it running with small changes in my DOS config.sys file.

This is a monumental software package, but if you intend to create Windows applications, this is the software you should use to do it.

Inside Visual C++, David J. Kruglinski (Microsoft Press, Redmond, Wash., 1993, 631 pp. plus diskette; $39.95)

Early press releases referred to this book by the cumbersome but helpful title *The Microsoft Guide to C++ Programming in Windows*. The current title misrepresents the book a little, since it doesn't contain much information about the inner workings of Visual C++, while it does say a lot about C++ programming for Windows with the foundation class library. It doesn't replace Petzold's book, but a synthesis of the two will surely be forthcoming.

Kruglinski recognizes that programmers have purchased far more C++ books than they have read. He includes a 32-page summary of what he considers the essentials of C++ that you need to understand to read this book. I thought it was good, but his editors made him call it "a personal view."

I think the most useful thing about the book, justifying its title, is the way it leads you through all of the ins and outs of using Visual C++. If you want to go beyond the Visual C++ tutorial, or if you just find 40 pounds of manuals daunting, get this book and use it as your guide.

Visual Basic 3.0, Professional Edition (Microsoft, Redmond, Wash.; $495)

Basic began as a limited teaching language in the 1960s. In the mid 1970s and early 1980s, Microsoft's version of Basic became a nearly universal language for personal computers. It led to a proliferation of amateur programs. These programs arose quickly and easily from user needs. Small Basic programs are easy to write and understand. However, they tend to grow until they become unmaintainable. Basic is now largely obsolete as a programming language, but Microsoft uses it for special purposes, such as the macro language for Word for Windows.

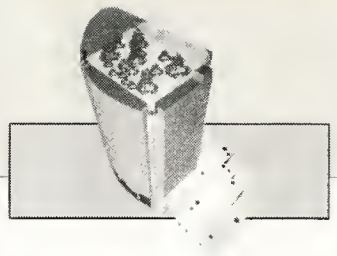
The event-driven structure that underlies Windows leads to the need for a large number of small routines. Visual Basic takes advantage of this fact. Users construct the application interface through direct manipulation of graphic elements, like buttons and menus, then associate Basic programs with them. Each event that an object needs to handle gives rise to a Basic subroutine. The detailed functionality is in the small programs, while the system retains control of the interface elements that the user specified.

As described, Visual Basic sounds like a prototyping tool or a vehicle for relatively limited programs. However, Microsoft has made it possible for you to build powerful systems with Visual Basic. You organize Visual Basic applications into projects. The graphic elements with associated Basic programs are called forms. In addition to forms, you can include code modules. These allow you to build subroutines that you can call from more than one form.

Of course, the code modules are an open invitation to the kind of amateur programming described earlier. Recognizing this, Microsoft has provided a number of improvements to Basic to help you avoid the kind of spaghetti code and global scopes of original Basic. The most important drawbacks of original Basic—line numbers and the GOSUB command—are long gone. Instead, you can use the same kinds of control structures that you have in other languages. You can even declare a procedure to be private to its code module. If you're a real purist, you can tell Basic to require you to declare all variables before you use them.

Another way to extend the power of Visual Basic is to link your program to procedures in dynamically linked libraries (DLLs). You can write your own DLL procedures—presumably in C, C++, or assembly language—or you can use any Windows system DLL pro-

continued on p. 99



Guest Editor's Introduction: **Toward a World Filled with Computers**

Ken Sakamura

University of Tokyo

It is my pleasure to present another East Asia issue, the first in two years. In my Guest Editor's Introduction of 1991[1] I noted some differences between the North American and Japanese microelectronics research and development scenes. The United States tended to emphasize high-powered applications such as superfast workstations and gigabit networks. In contrast, Japan gave more attention to the need for cost performance in home electronics and other consumer-oriented products.

That was two years ago. Today, however, that clear-cut distinction is changing. True, we still see continued competition over development of high-performance CPUs in the 100-MIPS-plus class for workstations and personal computers, with little regard for power consumption or chip size. At the same time, however, growing importance is being given around the world to microprocessors in the 10-MIPS range. These processors—which along with high performance feature low-power dissipation and small chip size—support embedded systems. We are seeing the appearance of high MIPS-per-watt microprocessors. Ever since the collapse of the Berlin Wall between East and West, emphasis has been shifting away from military technology toward civilian electronics markets. Even the United States is pouring new energy into embedded processors for portable systems and other consumer products. The applications for high-performance microprocessors are beginning to change dramatically.

In the TRON Project that we have been promoting over the last several years, we have continued to describe the future computer society as evolving along certain lines. We see the objects that surround us in our daily lives becoming increasingly embedded with computer chips, sensors, and actuators. These computerized objects then become linked by wired and wireless networks, forming distributed-processing systems in which the objects collaborate with each other.[2] (See Figure 1.)

We are now well into the 1990s, and things are moving increasingly in the direction of our predicted scenario. Recently, terms like *computer-augmented environment* and *ubiquitous computing*[3] seem to have become buzzwords in computer science. They describe an environment in which computers are used everywhere around us. This trend reflects the advances in development of the constituent technologies, such as small but high-performance sensors and displays, and high-performance sensors with low-power dissipation. It is becoming quite feasible to realize a world filled with computers.

In the next few years the world of microelectronics is likely to undergo great changes. The time has come to look ahead to applications for a world full of computers. With this background in mind, I describe recent microprocessor developments in Japan and, as in past issues, bring readers up to date on the TRON Project.

Japan's microprocessors

It is still true that most originally developed processors in Japan are designed for embedded system use. Japan has a number of general semiconductor manufacturers, including NEC, Toshiba, Hitachi, Fujitsu, and Mitsubishi Electric. But when it comes to microprocessors for workstations and personal computers, the manufacturers are limited to licensed production of the architectures of US companies (especially RISC chips). In view of the demand for RISC chips, none of the Japanese firms is developing its own original chip for

0272-1732/93/1000-0006$03.00 © 1993 IEEE

workstation or personal computer use. Moreover, they are hesitant to attempt developing chips compatible with the Intel x86 family because of the potential for legal trouble. The result is a further strengthening of the trend toward chips for embedded systems.

A boom in CPU development for low-power equipment persuaded Japanese manufacturers to develop original-architecture microprocessors (MPUs) and microcontrollers (MCUs). But up to now these were nearly all 4-, 8-, or 16-bit products. (In terms of quantity the 4-bit chips are most common.) Practically the only 32-bit chips to date have been the TRON-specification chips developed by six firms and the NEC V series.

Since last year however, 32-bit MPUs and MCUs aimed at large-scale use in portable systems have increased. ARM and Hobbit are well known outside Japan. Since 1992 various firms in Japan have announced 32-bit chips with low-power consumption aimed at the portable equipment market. The chips typically perform at 10 to 20 MIPS (in terms of VAX MIPS; that is, VAX 11/780 performance on Dhrystone 1.1 or 2.1 benchmarks). Their power dissipates at under 500 mW; the CPU core size is less than 50 mm^2. Specific applications include highly portable personal information systems and game machines.

The instruction set architecture of these chips typically blends RISC and CISC approaches, and a 16-bit instruction format improves object code efficiency. In other words, rather than going all out for performance as a pure RISC, Japan's manufacturers balance performance needs with the needs for small object code size and higher code density. Architecturally, the design is conservative, but applications of state-of-the-art technology realize the advantages of both RISC and CISC approaches.

It is this reworking of processor design that deserves our attention. Because of the emphasis on preserving past computer system investment, the companies have long been reluctant to depart from architectures like the IBM 370 or Intel x86. But in the case of processors for embedded systems, they have found it relatively easy to switch over to a new microprocessor design each time new equipment is introduced. When they compare the new designs to larger computer systems, they are less concerned about past investment. Makers of VCRs or copiers, for example, who are embedded-MPU users, care little about whether past software investment or yesterday's object code will run on a new system. Their concern is for deriving the maximum cost performance available at a given point in time. Thus they can readily adopt a processor with a new architecture.

In the latter half of 1992, NEC's V800 series and Hitachi's SH7000 series were announced as low-power 32-bit microprocessors. Then in the first half of 1993, Toshiba's TX2 and Mitsubishi's M16 were announced as second-generation TRON-specification, low-power-consumption processors for embedded systems. Common to each of these products is

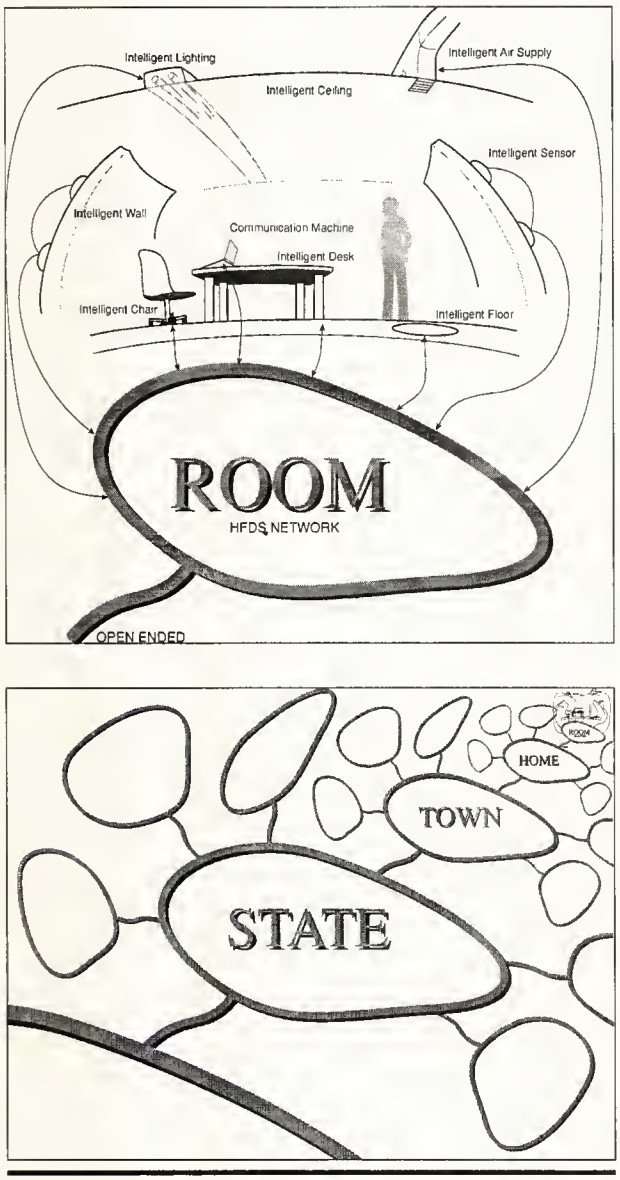


Figure 1. Intelligent objects and their networks.

that their manufacturers are making available ITRON-specification real-time operating systems. Table 1, next page, outlines the features of each new processor.

High-performance microprocessor development continues as well. Japanese manufacturers are presently gearing up to market three high-performance processors with original designs. Currently under development are Hitachi's Gmicro/500, Fujitsu's μVP, and Mitsubishi's Gmicro/400. The last of these is a high-speed, 32-bit processor for embedded systems that supports only integer calculations.

Table 1. The 32-bit, low-power-consumption processors in Japan.

Processor	Features
NEC V810	32-bit, low-power-consumption microprocessor 2.2V to 5V power supply; 15-MIPS performance at 5V and 25 MHz with 500-mW power dissipation and 40-mW power consumption at 2.2V and 10 MHz 32 general registers of 32 bits each; support for both 16-bit and 32-bit instruction formats, for efficient object code density Bit-string manipulation instructions; single-precision floating-point operation instructions conforming to IEEE Std. 754 0.8-µm CMOS technology; 240,000 transistors on a 7.7×7.7-mm chip; an on-chip 1-Kbyte instruction cache RX732 ITRON-specification real-time operating system 16-bit external bus version (V805) available
Hitachi SH7000 series	32-bit microcontroller family with on-chip DSP facility 16-MIPS performance at 20 MHz with a 5V power supply On-chip, 16-bit hardware multiplier unit for (16 bits × 16 bits + 42 bits) 42-bit multiplication and accumulation operation performed in 100 ns to 150 ns 500-mW power consumption, or 100 mW with a 3V power supply 0.8-µm CMOS process with a CPU core integrating 38,500 transistors in an area of 6.58 mm^2 Model SH7032 with peripheral functions and on-chip memory integrates approximately 593,000 elements on a 10.78 ×10.1-mm chip 16 general registers; 16-bit fixed-instruction format µITRON-specification operating system available
Toshiba TX2	32-bit, TRON-specification, low-power-consumption microprocessor Approximately 14-MIPS performance at 25 MHz (5V power supply), 1.7 times faster than the first-generation TX1 1/10th power consumption cut during WAIT instruction execution 16-bit instruction execution in one clock cycle Chip size one half that of the TX1
Mitsubishi M16	TRON-specification microcontroller with 16-bit external bus and 32-bit internal bus 4- to 5-MIPS average performance at 10-MHz operation (5V power supply) 16-bit instruction execution in one clock cycle Typical configuration (M31000S2FP) integrates 2 Kbytes of memory and peripheral functions

These are the only general-purpose microprocessors with an original architecture. As mentioned earlier, other products are being developed that license the RISC architectures of US firms (Sparc, HP-PA, Alpha, MIPS).

Hitachi Gmicro/500. This 32-bit, TRON-specification microprocessor adopts a superscalar architecture that permits 130-MIPS performance at 66 MHz. When compared to Intel's 66-MHz Pentium, this chip dissipates only a third as much power, is a third smaller, and is 20 percent faster. Its designers used a 0.6-µm CMOS process.

How is it that the Gmicro/500 boasts performance surpassing Intel's Pentium? The answer lies in its being targeted mainly for embedded system use. Because of Intel's lock on the personal computer market, the Gmicro/500 had no other choice but to aim for use in embedded control systems. And the only way to win in the high-performance embedded system market was to focus on small chip size, low-power consumption, and high speed. In other words, its excellence is the result of its being aimed not at personal computer and workstation use but at embedded systems.

This microprocessor also, of course, runs an ITRON-specification operating system. In addition, designers are developing a BTRON2-specification operating system for workstation use. The latter operating system supports a hypertext structure at the operating system level, as well as offering functions for multimedia support, multilingual processing, and distributed processing; it provides a compact kernel.

Fujitsu μVP. The μVP is a vector-processing-architecture coprocessor for direct connection to TRON-specification 32-bit microprocessors. At 50 MHz, 206-Mflops single-precision speed, and 106-Mflops double-precision speed, this 0.5-μm CMOS chip performs on a par with early supercomputers.

Separate articles on these two products appear elsewhere in this special issue. They are examples of how manufacturers continue to take up the challenge of developing a Japanese-original chip running an originally developed operating system, and are steadily making progress in implementing the goal.

There was a time when people thought the future belonged only to RISC and that CISC chips would disappear. But in Japan the emphasis is on object efficiency. For this reason a blend of RISC and CISC designs was sought. As a result, people have come to appreciate an architecture like that of the TRON-specification chips; new microprocessors implementing this architecture are being developed today. Some of the applications for which they are being used are high-performance numerical control machines and communication control processors. Then we have the μVP coprocessor with its vector-processing approach, a unique product that stands out among the peripheral chips designed for the TRON architecture. A single-board computer running the Gmicro/500 and μVP is said to achieve performance equivalent to that of a Cray 1 supercomputer, demonstrating amazing progress in microelectronics technology.

TRON Project update

The TRON Project is looking ahead to a world filled with computers. As I noted at the beginning of this piece, interest is beginning to focus on environments that are filled with computers everywhere you look. The TRON Project has constructed a pilot TRON-concept Intelligent House[4,5] with around 1,000 built-in computer elements. We are about to begin construction of the first TRON-concept Intelligent Building that incorporates tens of thousands of computers.

What sets the TRON Project apart is its attempt to get a jump on the next computer age by considering what kinds of computer applications are likely to emerge, actually building such applications, and feeding back the results into the design of basic components such as microprocessors and operating systems.

Besides trying to determine the most suitable architecture for an age when the number of microchips in use is thousands or ten thousands of times greater than today, this project takes a comprehensive look at questions such as the following. What should computers be made to do? What should they not be made to do? What kinds of infrastructures are needed? What rules are necessary for data interchange? In these ways the project is planning and building information infrastructures for the future.[6]

From the TRON Intelligent House to the TRON Hyper-Intelligent Building. We completed the TRON-concept Intelligent House experiment, conducted as an application project to get an advance look at the future, in the spring of 1993. At the same time we finished the basic design of the TRON Hyper-Intelligent Building, incorporating tens of thousands of computer elements, in preparation for the start of construction later this year. (See Figure 2, next page.)

The significance of this project is that a life-size model of a "computers everywhere" building will be built and put to actual use as a place of work. The building's computers will be able to locate people wherever they go and will fine-tune lighting (see Figure 3), temperature, and other environmental factors to personal preferences. The overall model is that described earlier of "intelligent objects" (ordinary objects containing microchips, sensors, and actuators) linked in wired or wireless networks that enable them to coordinate their actions. The component parts making up these networks are the results of fundamental research and development taking place in the TRON Project. In addition to the microprocessors[7] used to control the intelligent objects, these important development results include real-time operating systems, high-level data interchange protocol, and human-machine interface specifications.

The importance of open architecture. One of the vital requirements of a processor in intelligent objects is outstanding real-time performance. Important likewise are the ability to be used as an ASIC core, the possibility of applying the same architecture to a whole range of processors from low-power-consumption models to high-end products, and the adoption of an open-architecture policy so that anyone is free to develop compatible microprocessors. When intelligent objects become networked in the tens of thousands or millions, it would be highly unlikely that all the component parts could be supplied by one corporation. In such an age, the basic architectures, operating system interfaces, data interchange protocols, and other basic technologies will have to be open to all as social infrastructure. Success is not possible under the dominance of any one company. I believe this open-system approach needs thoughtful consideration.

Subprojects. I have already touched on the status of the TRON-specification microprocessors. Other fundamental subprojects are currently under way.

ITRON. The ITRON-specification real-time operating systems for embedded systems have been implemented for most Japanese microcontrollers, and it has become a de facto industry standard. The ITRON standards continually undergo

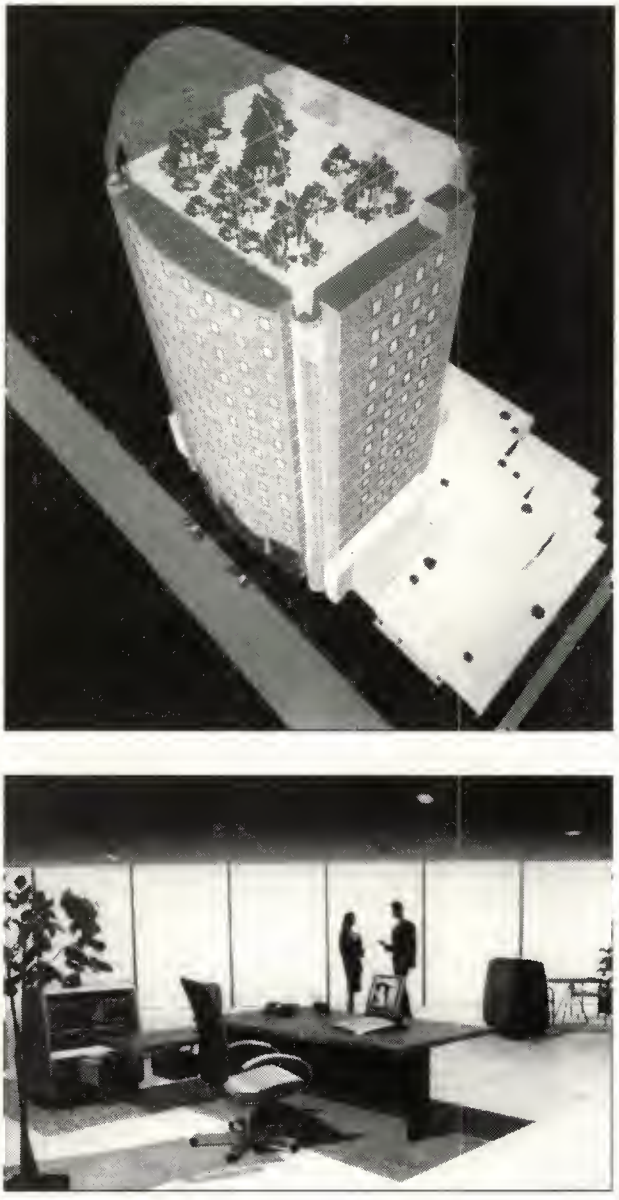


Figure 2. The TRON Hyper-Intelligent Building. © *Ken Sakamura 1990.*

improvements; the newest version called μITRON 3.0 offers network support. That is, we extended the specification to permit application to distributed systems connected in loosely coupled networks. When CPU nodes running μITRON 3.0 are networked, programmers can use ordinary system calls to manipulate tasks or semaphores in other nodes. Naturally, the specifications are open. Any interested readers can obtain copies of the English-language specifications via Internet's

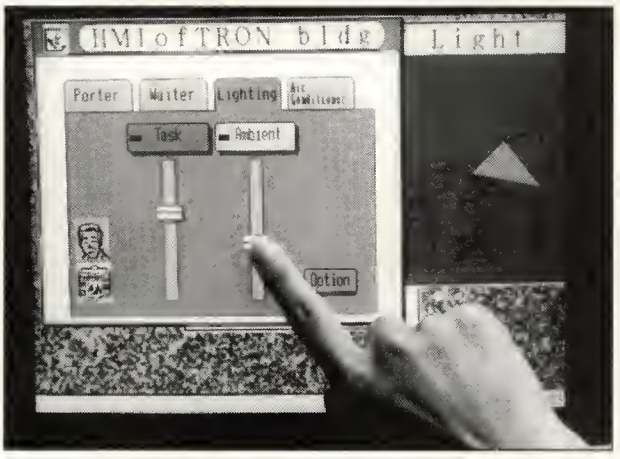


Figure 3. A BTRON screen panel for lighting control. © *Ken Sakamura 1990.*

anonymous ftp utsun.s.u-tokyo.ac.jp; the directory is /TRON/ITRON/SPEC.

BTRON. When the ubiquitous computer age finally arrives, failure to standardize a human-machine interface (HMI) will invite confusion. The BTRON subproject is not limited to personal computers but deals with this interface in a much broader sense. The BTRON operating system specifications define a compact, highly efficient graphical user interface-based operating system designed for use even in light switches, of the kind adopted in the TRON-concept Intelligent House (see Figure 1). Compared to Microsoft Windows, it runs at high speed with minimal resources. (A workable system has been implemented around an i286 processor running at 16 MHz, with 4 Mbytes of memory and a floppy-disk drive.)

BTRON's fully multitasking operating system also supports a hypertext/hypermedia function at the system level. Since it would hardly be feasible to apply the GUI approach to all the switches around us, the TRON HMI specifications[8] also describe non-GUIs, such as physical switches and handles (called SUI). Also specified in the BTRON subproject is the TAD (TRON application databus) high-level data interchange protocol for use among intelligent objects. This protocol is starting to be used in systems consisting of intelligent objects linked by a simple token-ring bus (a μBTRON-specification bus[9]), for system operation and the like, to achieve a consistent operation environment.

CTRON. These specifications apply to operating systems used in communications systems and servers such as big telephone companies and private branch exchanges. In 1993 Tandem Computers announced a nonstop computer system running on a CTRON-based operating system.

TRON progress. As this summary should indicate, the TRON Project is proceeding at a steady pace, making its mark

as an original Japanese computer development project. The standards and specifications produced by the TRON Project are openly available to anyone by writing to TRON Association, Katsuta Building 5F, 3-39, Mita 1-chome, Minato-ku, Tokyo 108, Japan. (The contact address for North America is TRON Association US Liaison Office, PO Box 23990, Tempe, AZ 85285.) Moreover, the standards can be used freely, without licensing fees.

SINCE READERS ARE ESPECIALLY INTERESTED in microprocessors, I have prepared articles in this special East Asia issue on two of the chips I've mentioned: the general-purpose microprocessor Gmicro/500 and the "single-chip supercomputer" μVP coprocessor. Many other unusual microprocessors are being developed in Japan, but of these I have chosen to present a high-speed fuzzy chip. I hope you enjoy reading these three articles.

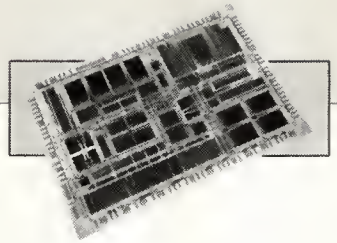
References

1. Ken Sakamura, "Guest Editor's Introduction," *IEEE Micro*, Aug. 1991, pp. 12-15.
2. Ken Sakamura, "The Objectives of the TRON Project," *Proc. Third TRON Project Symp.*, Springer-Verlag, Tokyo, 1987, pp. 3-16.
3. Special Issue: Computer Augmented Environments, *Comm. ACM*, July 1993.
4. Ken Sakamura, "About the Cover: The TRON Intelligent House," *IEEE Micro*, Apr. 1990, pp. 6-7.
5. Volker Hartkopt et al., *Designing the Office of the Future: The Japanese Approach to Tomorrow's Workplace*, John Wiley & Sons, New York, 1993.
6. Ken Sakamura, "TRON Application Projects: Gearing Up for HFDS," *Proc. Eighth TRON Project Symp.*, IEEE Computer Society Press, Los Alamitos, Calif., 1991, pp. 2-14.
7. Ken Sakamura, *An Introduction to T-Open*, TRON Association, Tokyo, 1992.
8. TRON Electronic Equipment HMI Research Group, *TRON Human-Machine Interface Specifications*, TRON Association, 1993.
9. Ken Sakamura et al., "The μBTRON Bus: Functions and Applications," *Proc. Third TRON Project Symp.*, Springer-Verlag, 1989, pp. 101-112.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 150 Medium 151 High 153



The Gmicro/500 Superscalar Microprocessor with Branch Buffers

Featuring a RISC-like dual-pipeline structure for high-speed execution of basic instructions, the Gmicro/500 represents a significant advance for the TRON architecture. Upwardly object-compatible with earlier members of the Gmicro series, this microprocessor uses resident dedicated branch buffers to greatly enhance branch instruction execution speed. Also, its microprograms simultaneously employ dual execution blocks to effectively execute high-level language instructions. Fabricated with a 0.6-μm CMOS technology on a 10.9 × 16-mm die, the chip operates at 50/66 MHz and achieves a processing rate of 100/132 MIPS.

Kunio Uchiyama

Fumio Arakawa

Susumu Narita

Hirokazu Aoki

Ikuya Kawasaki

Shigezumi Matsui

Mitsuyoshi Yamamoto

Norio Nakagawa

Ikuo Kudo

Hitachi, Ltd.

As one of the new high-end microprocessors that perform at over 100 million instructions per second,[1-4] the Gmicro/500 builds upon the well-established TRON architecture.[5] The TRON specification includes 135 variable-length instructions with 14 addressing modes. Operating at 50/66 MHz, this chip is upwardly object-code compatible with earlier members of the Gmicro series.[6-9] To achieve a processing rate of 100/132 MIPS, it uses a sophisticated superscalar technique and high-speed branch accelerators optimized to a complex instruction-set computer architecture.[10, 11]

We established the performance target for the Gmicro/500 at over 100 MIPS with two constraints in mind: object-code compatibility and cost. The Gmicro microprocessors already have many customers in such diverse fields as industrial equipment, communication switching systems, fault-tolerant computers, and business workstations. To retain this existing customer base, the Gmicro/500 must support the TRON architecture.

This well-established architecture features variable-length instructions and includes rich, high-level instructions, such as string, variable-length bit field, multiple-word load/store, and operating system-related instructions. We developed a new superscalar method for this CISC architecture that can execute two instructions simultaneously. Our technique can also effectively process high-level instructions using two pipelines.

In microprocessor design, cost depends on clock frequency and power dissipation. A high clock frequency increases power dissipation, which in turn necessitates costly large-scale integration packaging and expensive system cooling. A high clock frequency also requires high-speed external memory, further increasing system costs.

To improve performance without relying solely on an increased clock frequency, we sought to reduce the effective CPI, or clocks per instruction, to 0.5. In other words, we wanted our system to execute two instructions per clock. To achieve this goal, we needed to develop not only a sophisticated superscalar technique but also high-speed branch accelerators that reduce branch penalties.

In introducing the Gmicro/500, we first describe the chip's specifications and internal architecture. Included with this is a discussion of the new sophisticated superscalar technique and high-speed branch accelerators that are optimized to a CISC architecture. Afterwards we will present the results of a performance analysis on this new microprocessor.

0272-1732/93/1000-0012$03.00 © 1993 IEEE

Table 1. Gmicro/500 specifications.

Parameter	Specification
Clock frequency	50/66 MHz
Performance	100 MIPS (50 MHz) 132 MIPS (66 MHz)
Basic CPI	0.5 clocks (two instructions per second)
Voltage	5 V
Power dissipation	7 W (50 MHz), 9 W (66 MHz)
Package	Ceramic QFP, 256 pins
Technology	0.6-µm CMOS, three metal layers
Die size	10.9 × 16.0 mm
Instruction cache	8 Kbyte, 2 way
Operand cache	8 Kbyte, 2 way, write through
Instruction TLB	64 entries, 2 way
Operand TLB	64 entries, 2 way
Branch target buffer	64 entries, 2 way
Return buffer	8 entries
Store buffer	4 entries by 8 bytes
External address bus	32 bits
External data bus	64 bits

Chip specifications

Table 1 summarizes the Gmicro/500 specifications. We used a three-metal-layer, 0.6-µm complementary metal-oxide semiconductor technology for its fabrication. Shown in Figure 1, this microprocessor integrates 1.65 million transistors on a 10.9 × 16-mm die. The chip incorporates a RISC-like superscalar integer execution unit, resident two-way set-associative 8-Kbyte instruction and operand caches, and a 64-entry branch target buffer.

The Gmicro/500 also has an eight-entry return buffer, a four-entry store buffer, and a memory management unit with comprehensive section and page protection. Its floating-point processing unit—fully compatible with the ANSI/IEEE 754-1985 standard—operates in parallel with the integer execution unit. As shown in Figure 2, the Gmicro/500 achieved 100/132 MIPS at 50/66 MHz—the top performance of recent CISC microprocessors.

Internal architecture

As shown in the block diagram of Figure 3 (next page), the Gmicro/500 consists of five basic units:

- Instruction
- Decode
- Execution
- Access
- Floating-point processing

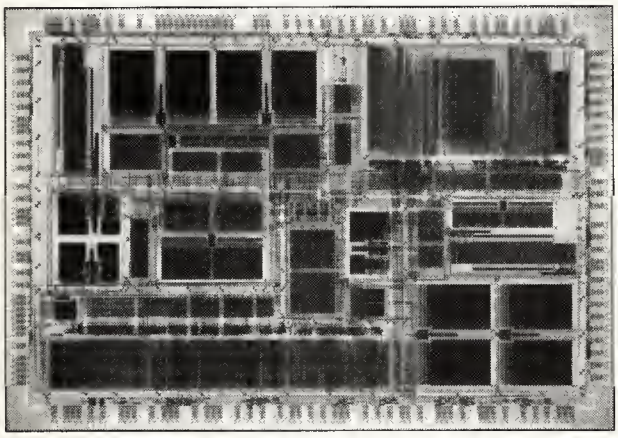


Figure 1. Photomicrograph of the Gmicro/500.

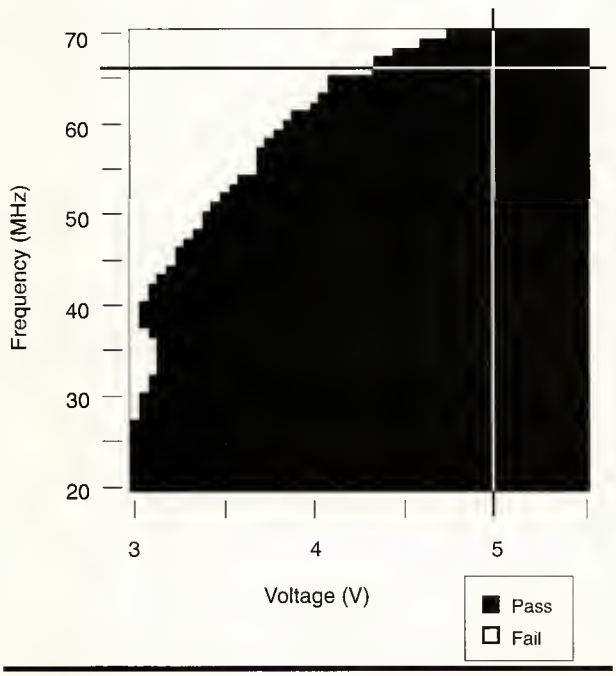


Figure 2. Schmoo plot.

Using the prefetch address generator, the instruction unit prefetches instructions from the instruction cache or external memory. These it transfers to the instruction prefetch queue. The prefetch queue consists of three sets of 64-bit latches. In addition to the 64-entry instruction translation look-aside buffer and 8-Kbyte instruction cache, the instruction unit contains a 64-entry branch target buffer and an eight-entry return buffer. These are dedicated to caching branch information for accelerated branch execution.

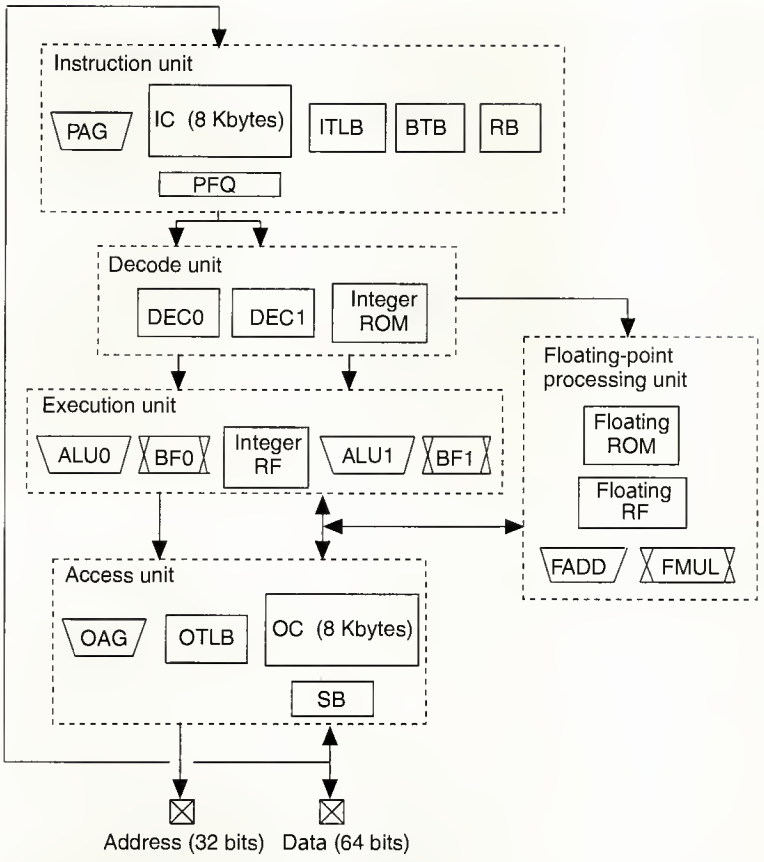


Figure 3. Gmicro/500 block diagram.

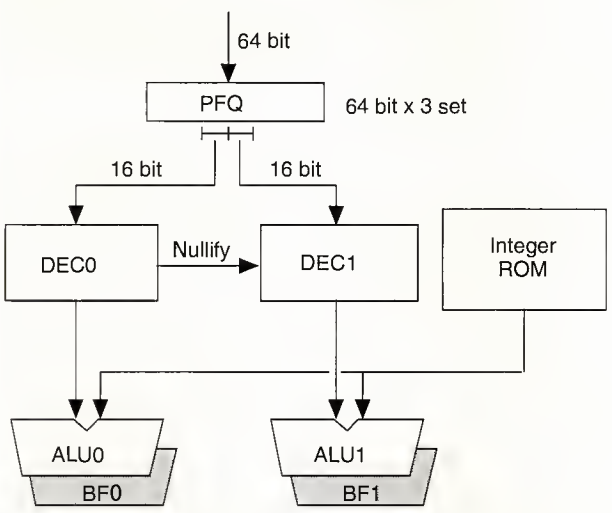


Figure 5. Decoding scheme.

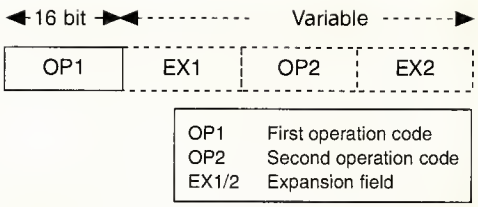


Figure 4. Instruction format for TRON specifications.

The decode unit decodes instructions using two 16-bit decoders (DEC0, DEC1). Because the length of instruction code can vary (see Figure 4), the simultaneous decoding of two instructions presents a major difficulty. The Gmicro/500 therefore decodes two instructions simultaneously only when the instruction decoded by DEC0 is 16 bits wide. As shown in Figure 5, DEC0 and DEC1 decode two successive 16-bit lengths that are extracted from the prefetch queue. If the information decoded by DEC0 is not a 16-bit-wide instruction, the result decoded by DEC1 is nullified.

In the Gmicro/500 instruction set, all basic instructions may be coded in a special 16-bit short format; compilers frequently use these short instructions. Table 2 lists the appearance probability of the field that follows the 16-bit first operation code (OP1) in five benchmark programs. The high probability of the OP1 field that follows the previous OP1 means that the simultaneous decoding by the two decoders has a high probability of succeeding. With high-percentage use of the 16-bit instruction format, the dual-pipeline decoding scheme significantly reduces implementation complexity without sacrificing performance.

Table 2. Appearance probability of the field following OP1.

Program	Type OP1	OP2	EX
Dhrystone	0.63	0.06	0.31
EDN-E	0.86	0.00	0.14
EDN-F	0.65	0.12	0.23
EDN-H	0.81	0.00	0.19
EDN-K	0.67	0.16	0.17
Average	0.72	0.07	0.21

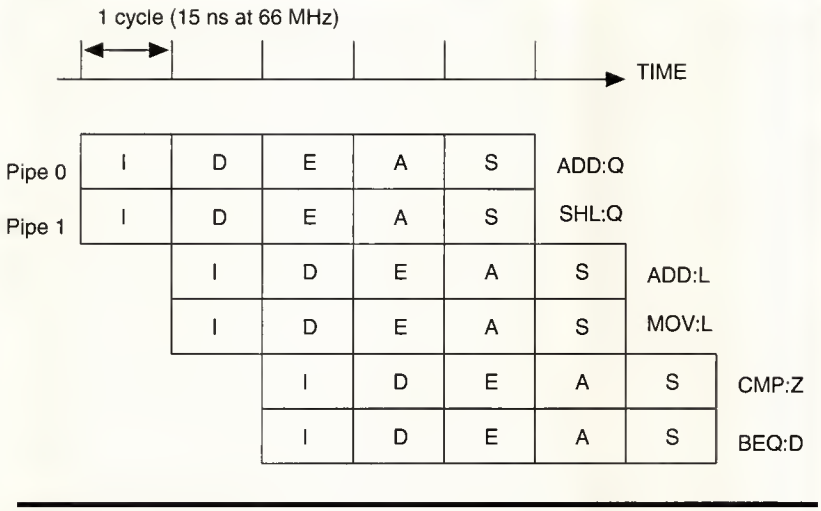


Figure 6. Dual pipelines.

Commonly used acronyms

ALU	Arithmetic logic unit
AU	Access unit
BRA	Branch always
BSR	Branch to subroutine
BTB	Branch target buffer
FADD	Floating-point adder
FILO	First in, last out
FMUL	Floating-point multiplier
IC	Instruction cache
ITLB	Instruction TLB
OAG	Operand address generator
OC	Operand cache
OTLB	Operand TLB
PAG	Prefetch address generator
PFQ	Prefetch queue
RB	Return buffer
RTS	Return from subroutine
SB	Store buffer
TLB	Translation look-aside buffer

The decode unit also includes an integer ROM that stores the microprograms for high-level instructions, such as string, variable-length bit field, and operating system-related instructions. One of the two decoders controls the first execution cycle of the high-level instruction, and the 120-bit microinstructions from the integer ROM control the next execution cycles. Simultaneous use of dual arithmetic logic units and dual barrel shifters in the execution unit in the microprograms enables the Gmicro/500 to execute high-level instructions quickly.

The execution unit contains the ALUs (ALU0, ALU1), the barrel shifters (BF0, BF1), and an integer register file that has four read and two write ports. Using these duplicated resources, the execution unit can execute two basic integer instructions at the same time. It can execute high-level instructions by using the dual ALUs to concatenate and manipulate data in 64-bit units. (A high-level instruction is one that executes operations that would normally require several basic machine-code instructions.)

The floating-point processing unit contains a 55-bit floating-point adder, a 66-bit floating-point multiplier, a floating-point ROM, and a floating register file with one write and two read ports. Because the floating ROM independently controls the floating-point processing unit, the processor can execute floating-point instructions in parallel with integer instructions.

The remaining unit of the Gmicro/500 is the access unit, used to control operand access and external bus cycles. It includes a 64-entry operand TLB, an 8-Kbyte operand cache that supports a write-through protocol, a four-entry store buffer, and an operand address generator. The access unit connects to the outside through a 32-bit address bus and a 64-bit data bus.

The block size of the instruction cache and operand cache is 32 bytes. The fast cache fill can be done by burst transfer at 6.4 bytes per cycle. To keep cache coherency in multiprocessor systems, the Gmicro/500 automatically snoops the external bus address and invalidates the hit blocks in the instruction cache and operand cache. Since dual-port RAMs are used at the tag sections of both the instruction and operand caches, the bus snoop access to the caches can occur simultaneously with operation of the normal cache access. Using the operand address generator in the access unit, the hardware in the chip automatically handles the TLB miss, enabling the Gmicro/500 to minimize the TLB miss overhead.

Superscalar pipelining

The architecture of the Gmicro/500 builds upon sophisticated superscalar techniques.

Dual pipeline. The Gmicro/500 has dual-integer pipelines. As Figure 6 shows, each of these is divided into five stages:

- Instruction prefetch (I)
- Instruction decode (D)
- Operation execution (E)
- Memory access (A)
- Register store (S)

This structure allows the simultaneous execution of two basic integer instructions. The RISC-like superscalar pipeline structure optimizes the execution speed of register-register and basic load/store operations. Other operations (such as

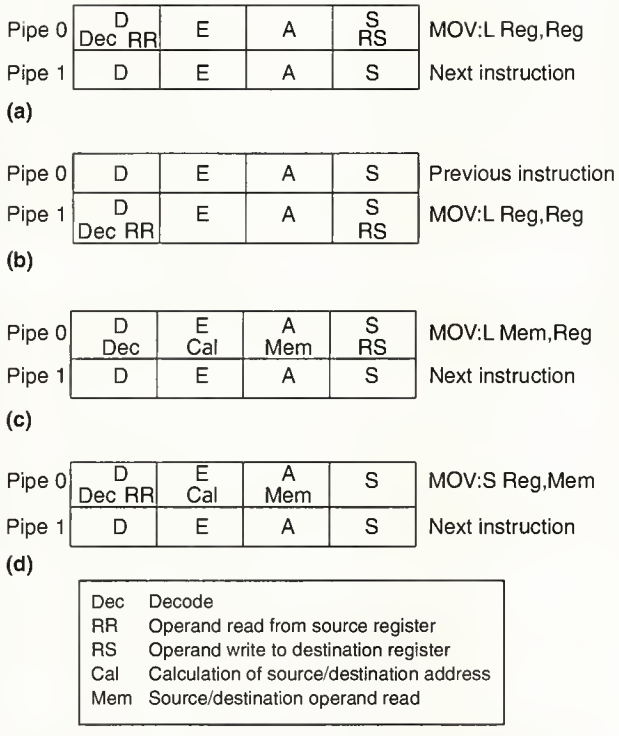


Figure 7. Pipeline stage of transfer instructions: MOV:L Reg,Reg on pipe 0 (a); MOV:L Reg,Reg on pipe 1 (b); MOV:L Mem,Reg on pipe 0 (c); MOV:S Reg,Mem on pipe 0 (d).

string and multiple-word load/store instructions) use the dual ALUs to increase execution speed by concatenating and operating on data in 64-bit units.

Figure 6 also shows a basic example of integer instruction execution flow. The pipeline streams pipe 0 and pipe 1 fetch ADD:Q and SHL:Q, respectively. The processor's decode unit decodes these two instructions simultaneously, then its instruction unit fetches the following ADD:L and MOV:L. When ADD:Q and SHL:Q are in the operation execution stage, the processor simultaneously decodes ADD:L and MOV:L, then fetches CMP:Z and BEQ:D. In this manner, the pipeline stages of 10 instructions run concurrently in five pipe-pair sequences every clock cycle, effectively completing two instructions per clock cycle.

Cache misses and register conflicts may cause insertion of wait cycles into the pipeline flow. Other obstacles (for example, branch operations) also pose potential hazards to the pipeline flow. Furthermore, high-level language instructions executed directly under microprogram control require a sequence of stages that extends over more than one pipe. The Gmicro/500 incorporates special features to minimize, and in some cases eliminate, obstacles to the pipeline flow, thus ensuring high performance for all types of programs.

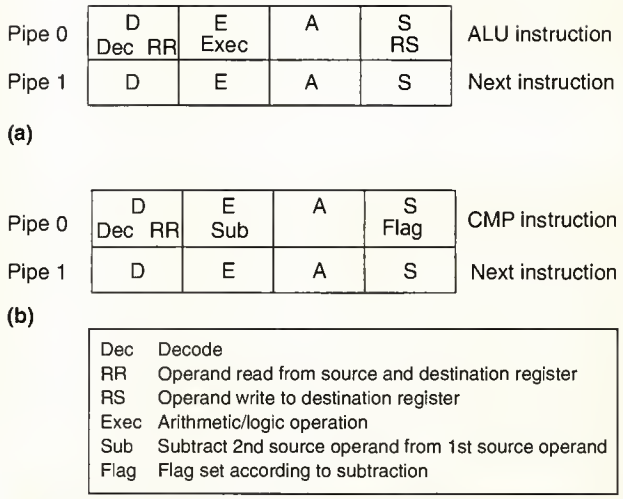


Figure 8. Execution of instructions: arithmetic/logic instructions (a); compare instructions (b).

Pipeline stage for transfer instructions. Figure 7 outlines the execution flow for transfer instructions. Register-register operations appear in Figure 7a and b. In stage D, the processor decodes the instruction and reads the source operand from the designated register. It then loads the source operand into the destination register in stage S, completing the transfer operation. Stages E and A are vacant—unused—in this type of operation. Figure 7a shows pipeline flow when the processor is executing MOV:L Reg,Reg operation on pipe 0, allowing simultaneous execution of the next instruction on pipe 1. In Figure 7b, the MOV:L Reg,Reg instruction is executed on pipe 1 at the same time that the previous instruction is executed on pipe 0.

Figure 7c shows the execution of a load operation, with the MOV:L Mem,Reg instruction executed on pipe 0 and the next instruction simultaneously executed on pipe 1. In this example, the processor decodes MOV:L Mem,Reg instruction in stage D of pipe 0 and calculates the effective address of the source operand in stage E. It then reads the source operand in stage A and loads it into the destination register in stage S.

Figure 7d shows a store operation in which the processor executes the MOV:S Reg,Mem instruction on pipe 0 while simultaneously executing the next instruction on pipe 1. In stage D on pipe 0, the processor decodes the MOV:S Reg,Mem instruction and reads the source operand from the designated register. It calculates the effective address of the destination location in stage E and stores the source operand in the addressed memory location in stage A, completing the transfer operation. In this example, stage S is vacant (unused).

Execution of arithmetic/logic instructions. Figure 8a shows the execution flow of arithmetic/logic instructions. Here

the processor executes a register-register arithmetic/logic instruction on pipe 0. In stage D, it decodes the instruction, then reads the operands from the source register (or immediate data in the instruction code) and the destination register. It performs the arithmetic/logic operation on the operands in stage E, storing the result in the destination register in stage S (stage A is vacant).

In Figure 8b, the chip executes a compare instruction on pipe 0. In stage D, it decodes the compare instruction, then reads the operands from the first source register (or immediate data in the instruction code) and the second source register. The compare (arithmetic subtraction) operation takes place in stage E, and the result is reflected by updating the status flags in stage S, completing instruction execution (stage A is vacant). In both examples, the next instruction may execute simultaneously on pipe 1.

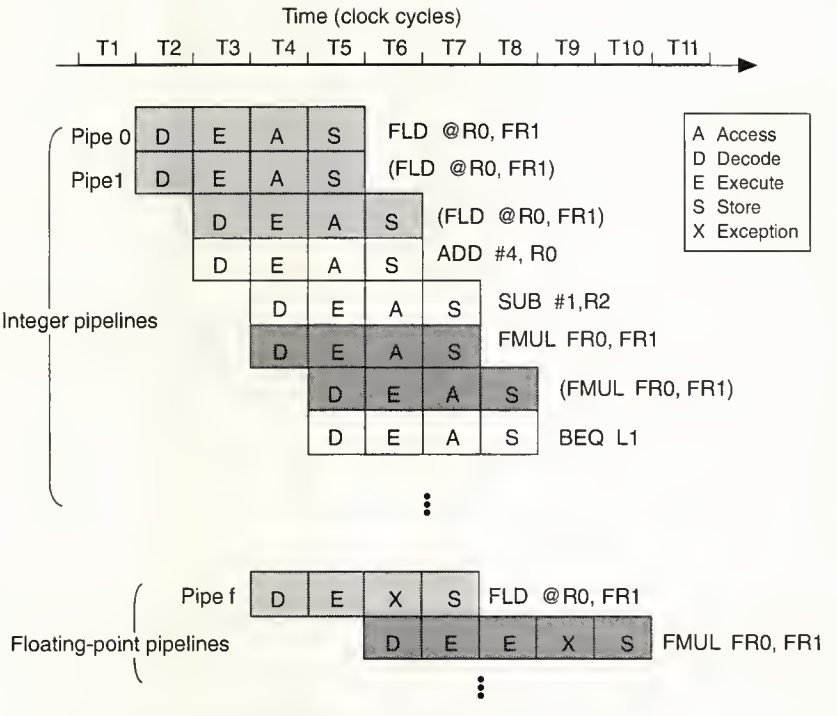


Figure 9. Execution of floating-point instructions.

Floating-point pipeline. The Gmicro/500 supports floating-point instructions that are fully compatible with the ANSI/IEEE 754-1985 standard. Figure 9 shows the execution of these instructions. The shaded parts are the pipes used for the floating-point instructions. The processor first decodes a floating-point instruction at stage D of integer pipe 0 or pipe 1, then sends it to the floating-point processing unit for the floating-point operation. The floating-point processing unit has an independent pipeline (pipe f) consisting of stages D, E, X, and S.

The floating-point processing unit reads the floating register file at stage D. At this stage it also reads the first microprogram, which controls the following stages, from the floating ROM. After executing the floating-point operation at each stage E, it checks the exception condition at stage X. Once the floating-point pipeline has started, the processor releases the integer pipelines so they can be used for the next instructions. Using three pipelines (pipes 0, 1, and f), the chip executes integer instructions and floating-point instructions in parallel. Consequently, it can effectively execute a variety of floating-point programs.

Bypass function. The Gmicro/500 executes most types of instructions in a single pipe, freeing the other pipe to allow simultaneous execution of a previous or subsequent instruction. This produces a basic instruction execution time of 0.5 cycles. There are cases, however, in which pipeline flow is obstructed, degrading overall performance. Figure 10 (next page) shows examples of pipeline obstructions (also called hazards), and also illustrates various bypass functions designed to minimize the effect of pipeline obstruction.

Figure 10a shows a load-use conflict. Here the MOV:L R0,R1 instruction attempts to access the contents of R0 before the previous load instruction (MOV:L Mem,R0) could update it with new data. To ensure data integrity, the Gmicro/500 effectively bypasses the new contents of R0 from stage A in pipe 0 of the MOV:L Mem,R0 instruction to stage E in pipe 1 of the MOV:L R0,R1 instruction. It is not necessary to wait until the previous instruction stores the destination operand.

Figure 10b shows a define-use conflict in two successive register-register operations. In pipe 0 of the initial pipe pair, the processor first reads the new contents of R0 (that is, the contents of R2) in stage D. Then it passes them from stage E of the MOV:L R2,R0 instruction to stage E in pipe 1 of the MOV:L R0,R1 instruction in the second pipe pair. This way, the processor can bypass new contents of R0 directly to MOV:L R0,R1 after accessing the source operand in MOV:L R2,R0. It does not have to wait for R0 to be updated in the MOV:L R2,R0 instruction. In general, potential data conflicts can be avoided if the compiler schedules instructions carefully.

Branch buffer

Branch instructions have usually been a major obstruction to pipeline flow. To minimize their effect on pipeline opera-

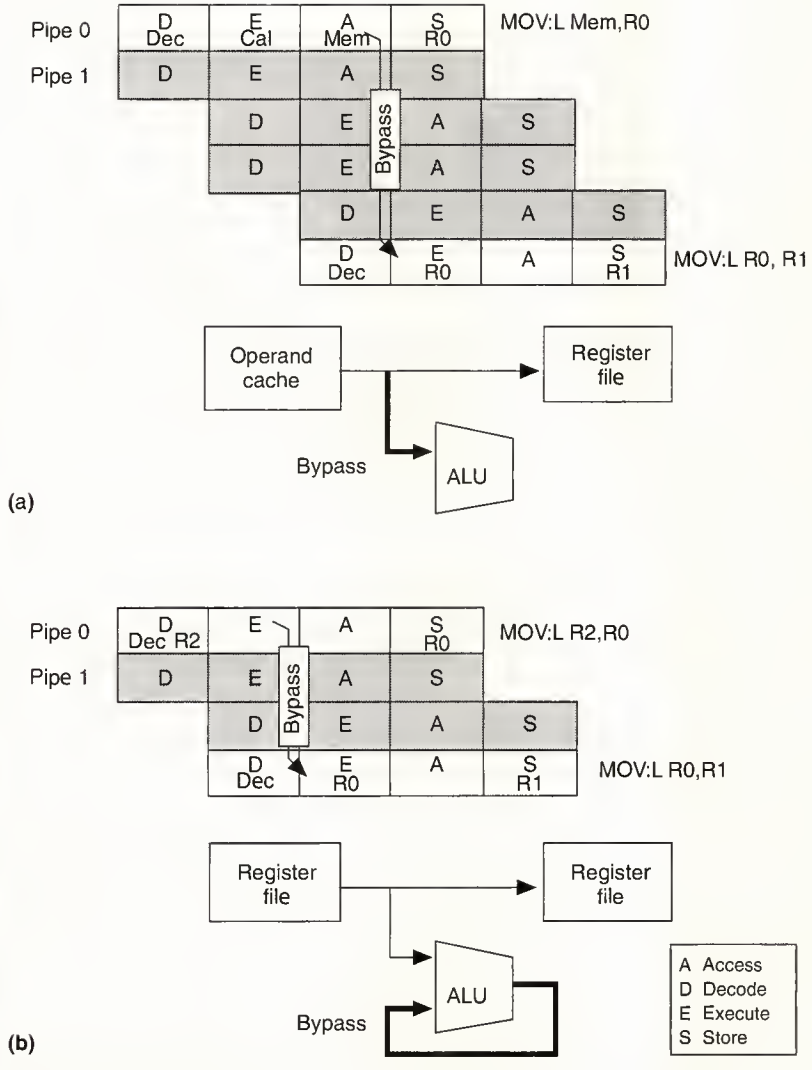


Figure 10. Bypass function: load-use conflict (a); define-use conflict (b).

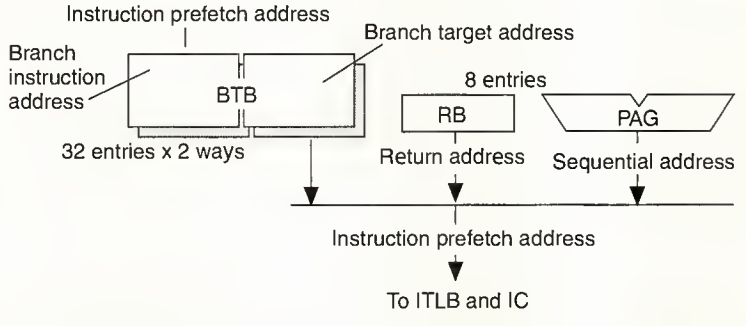


Figure 11. Structure of branch buffers.

tion, the Gmicro/500 features resident buffers dedicated to storing branch information. The branch target buffer, which has a two-way, set-associative structure, stores the addresses of branch instructions and each corresponding branch target address. This enables high-speed execution of branch-always and branch-to-subroutine instructions. The return buffer, which operates as a first-in, last-out memory, caches return addresses to allow high-speed execution of the return-from-subroutine instructions. Figure 11 shows the connection between the branch target buffer, the return buffer, and the prefetch address generator in the instruction unit. The processor selects the outputs from the branch target buffer, the return buffer, and the prefetch address generator. It then uses the selected logical address to access the instruction TLB and the instruction cache.

Zero-cycle branch. Use of the branch target buffer accelerates the execution of branch-always and branch-to-subroutine instructions. Figure 12 shows a zero-cycle branch in the branch-always execution. Figure 12a illustrates the effect on the pipeline when the processor executes a branch operation without the branch target buffer. The chip calculates the branch target address in stage E of the branch-always pipe. This address becomes the next instruction prefetch address (stage I in the next pipe), so the branch-always instruction requires a three-cycle penalty before executing the branch target instruction.

In Figure 12b, the chip effectively executes the branch-always instruction without having a pipe use the branch target buffer, thus allowing a zero-cycle branch. The instruction prefetch address of the branch-always instruction looks up the branch target buffer. It then reads the branch target address from the branch target buffer in case of a branch target buffer hit. The branch target address automatically prefetches the target instruction of the branch-always instruction, and the processor immediately executes the instruction residing at that location.

Fast return from subroutine. Using the return buffer accelerates the execution of the return-from-subroutine instruction. Figure 13a illustrates the effect on the pipe-

line when the chip executes a return operation without the return buffer. The processor pops the return address from the stack in stage A of the return-from-routine first pipe. It uses that address to fetch the target instruction of the return-from-routine through stage E of the last return-from-subroutine pipe. The return-from-subroutine instruction thus requires a five-cycle penalty before it executes the target instruction.

In Figure 13b, use of the return buffer reduces the penalty of the subroutine return from five cycles to two. In stage D of the return-from-subroutine instruction, the chip looks up the return buffer and reads the return address from the return buffer in case of a return-buffer hit. The return address automatically prefetches the return target instruction, and the chip immediately executes the instruction residing at that location.

Effect of branch target buffer and return buffer. Since the Gmicro/500 incorporates the branch target buffer and the return buffer, the compiler tries to make effective use of these resources. In the example of Figure 14 (next page), the compiler minimizes the number of taken conditional branches by introducing zero-cycle unconditional branches. In this way, it reduces the overhead of pipeline hazards caused by taken conditional branches. Figure 14a shows the nonoptimized case where the loop executes in four cycles because of the taken penalty of the conditional branch instruction.

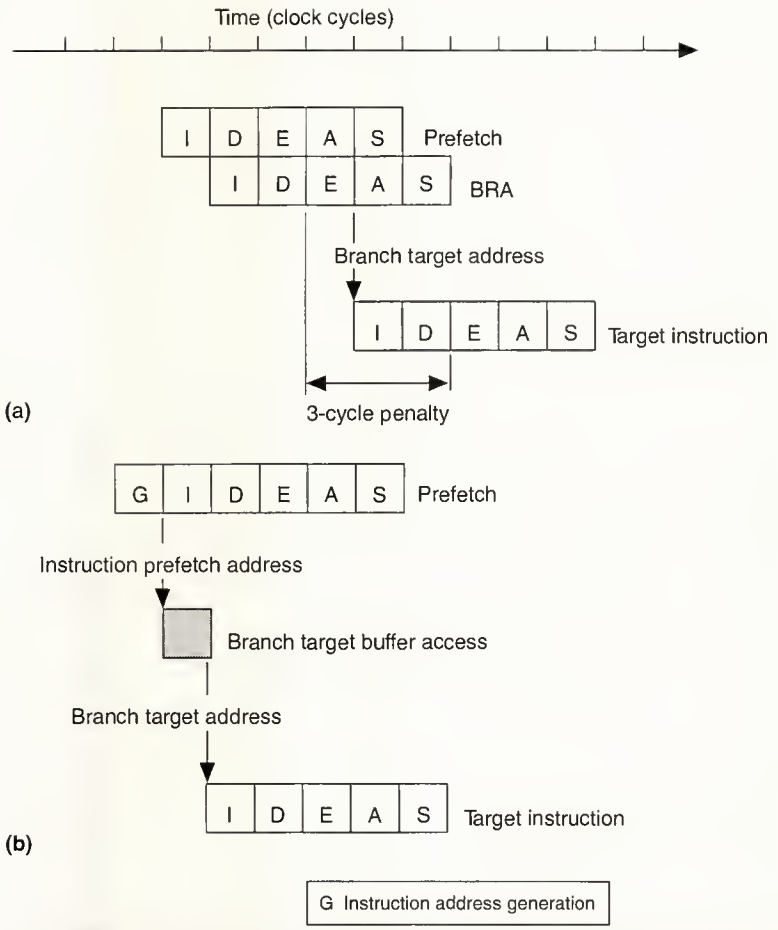


Figure 12. Zero-cycle branch: without branch transfer buffer (a); branch transfer buffer hit (b).

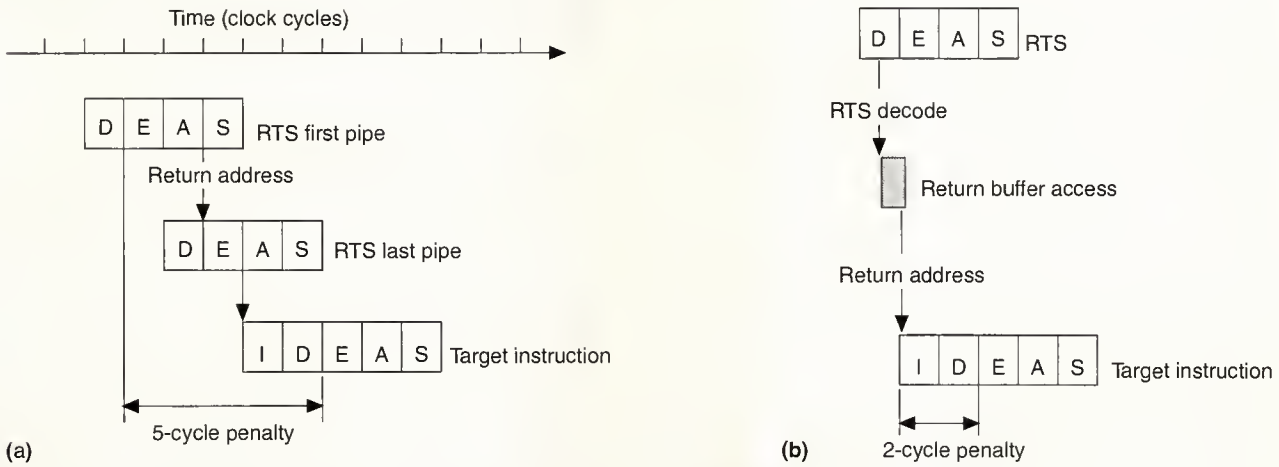


Figure 13. Fast return from subroutine: without return buffer (a); return-buffer hit (b).

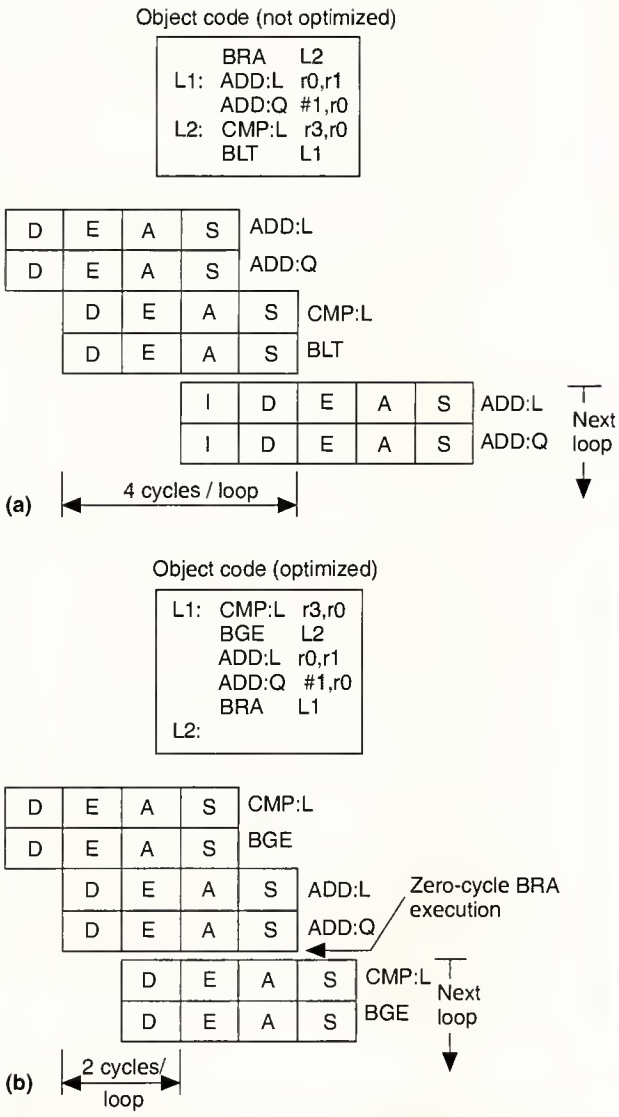


Figure 14. Branch optimization: nonoptimized (a); optimized (b).

Figure 14b demonstrates the optimized case where the loop executes in two cycles. Since the conditional branches are not taken during the loop in Figure 14b, no branch penalty occurs. The chip also uses the branch target buffer for zero-cycle execution of the branch-always instruction. Figure 15 shows the effect of the branch target buffer and the return buffer in the Dhrystone benchmark. Using the branch buffers effectively minimizes the execution time of branch-always, branch-to-subroutine, and return-from-subroutine instructions, reducing the total execution time to 73 percent.

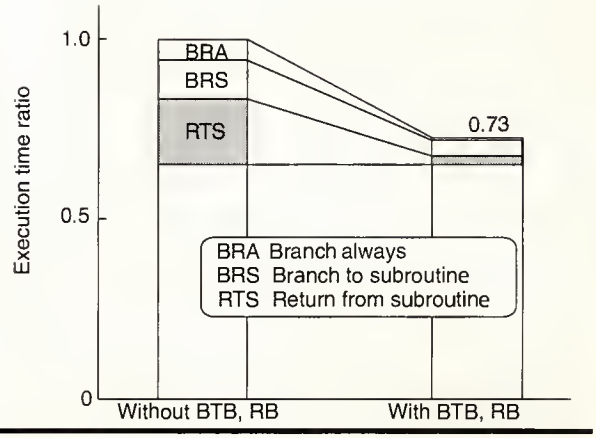


Figure 15. Effect of the branch transfer buffer and the return buffer.

Time	Pipe 0		Pipe 1	
72	ADD:Q	#imm,r5		
73			MOV:S	r5,r0
74	MOV:I	#imm:32,r2		
75	SHL:Q	#imm,r0	MOV:S	r5,r1
76	ADD:L	r0,r7	MUL:R	r2,r1
77				
78	MOV:S	r4,@(r7)	ADD:L	r6,r1
79	MOV:S	r5,r2	MOV:S	r4,@(d:16,,r7)
80	SHL:Q	#imm,r2	MOV:S	r5,@(d:16,r7)
81	ADD:L	r1,r2		
82			MOV:S	r2,r6
83	MOV:S	r5,@(r2)	ADD:Q	#imm,,r2
84	MOV:L	'@(d:16,r6),r1		
85			MOV:S	r5,@(r2)
86	ADD:Q	#imm,r1	MOV:L	'@(r7),r2
87	MOV:S	r1,@(d:16,r6)		

Figure 16. Parallel exception of dual pipelines.

Analysis of performance

Using the superscalar structure, branch buffers, and hardware-dependent optimizations for object-codes, the Gmicro/500 achieves high performance in a variety of programs. See Figure 16 for an example of dual-pipeline use. The code sequence used is part of the Dhrystone benchmark. The Gmicro/500 is executing two instructions simultaneously at times 75, 76, 78, 79, and so forth. The figure shows several pipeline hazards that prevent parallel execution. For example, register conflicts occur at times 72-73 and 81-82. The operand cache is conflicted at time 84-85. In this example, the chip executes two instructions simultaneously in 47 percent of cycles during this period despite various pipeline conflicts.

Figure 17 shows the results of dual-pipeline use and performance in Dhrystone Version 1.1 benchmark.[12] The graph shows the execution cycles of a Dhrystone loop. The Gmicro/500 can execute one Dhrystone loop in 285 cycles with the optimized object codes. This performance level is equivalent to 132 MIPS at 66 MHz (1 MIPS = 1,757 Dhrystone loops/s). The processor executes two instructions simultaneously on pipe 0 and pipe 1 in 65 cycles per loop. Using the dual ALUs and barrel shifters in the superscalar integer unit, the chip executes high-level instructions under microprogram control at a rate of 60 cycles per loop. There are 44 wait cycles in the loop caused by an operand interlock, a branch penalty, and so on. Pipe 1 is effectively used in 105 cycles in this Dhrystone loop, achieving a dual-pipeline performance gain of 27 percent (compared to 390 cycles to execute the loop with only one pipe in the chip).

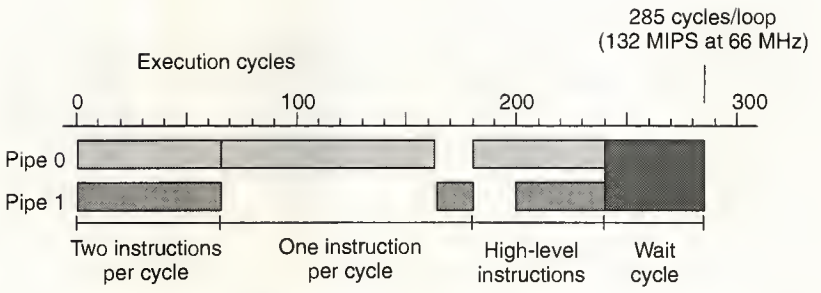


Figure 17. Dhrystone benchmark results.

The Gmicro/500 is upward object-code compatible with earlier members of the Gmicro family of processors, so software and other development tools such as the Gmicro/200 compiler can be used without modification. An optimized compiler for the Gmicro/500 based on the Gmicro/200 C compiler is currently under development.[13] To attain high performance, the optimized compiler adds functions that take full advantage of the Gmicro/500's unique features, such as its superscalar structure and branch buffers.

SYSTEMS USING THE GMICRO/500 ARE under development in such fields as communication switching systems and industrial equipment. Currently, we are evaluating the performance advantage of the superscalar pipeline, branch buffers, and multiprocessing functions under the real circumstances of operating systems and application programs. As we go, we will use the results of these evaluations to develop an even more powerful Gmicro processor.

Acknowledgments

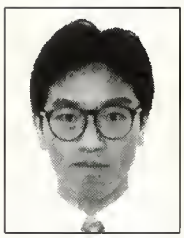
We thank Ken Sakamura of the University of Tokyo, as well as Hideo Inayoshi, Tadahiko Nishimukai, and the Gmicro staff members for their helpful suggestions. We also acknowledge all members of the Gmicro/500 development team for their valuable work.

References

1. D. Dobberpuhl et al., "A 200-MHz 64-bit Dual-Issue CMOS Microprocessor," *ISSCC Digest of Technical Papers*, Feb. 1992, pp. 106-107.
2. F. Abu-Nofal et al., "A Three-Million-Transistor Microprocessor," *ISSCC Digest of Technical Papers*, 1992, pp. 108-109.
3. J. Yetter et al., "A 100-MHz Superscalar PA-RISC CPU/Coprocessor Chip," *Symp. VLSI Circuits Digest of Technical Papers*, 1992, pp. 12-13.
4. J. Crawford et al., "The P5 Microarchitecture," *Fifth Ann. Microprocessor Forum*, 1992, pp. 11-1 to 11-16.
5. K. Sakamura, "Architecture of the TRON VLSI CPU," *IEEE Micro*, Vol. 7, No. 2, Apr. 1987, pp. 17-31.
6. T. Yoshida et al., "The Gmicro/100 32-bit Microprocessor," *IEEE Micro*, Vol. 11, No. 4, Aug. 1991, pp. 20-23, 62-72.
7. H. Inayoshi et al., "Realization of the Gmicro/200," *IEEE Micro*, Vol. 8, No. 2, Apr. 1988, pp. 12-21.
8. T. Kitahara and T. Satoh, "The Gmicro/300 32-Bit Microprocessor," *IEEE Micro*, Vol. 10, No. 3, June 1990, pp. 68-75.
9. T. Yoshida et al., "The Approach to Multiple Instruction Execution in the Gmicro/400 Processor," *Proc. Eighth Int'l Symp. TRON Project*, IEEE Computer Society Press, Los Alamitos, Calif.,1991, pp. 185-195.
10. S. Matsui et al., "Gmicro/500 Microprocessor: Pipeline Structure of Superscalar Architecture," *Proc. Ninth Int'l Symp. TRON Project*, IEEE CS Press, 1992, pp. 56-62.
11. F. Arakawa et al., "A CMOS 50-MHz CISC Superscalar Microprocessor," *Symp. VLSI Circuits Digest of Technical Papers*, 1993, pp. 11-12.
12. R.P. Weicker, "Dhrystone: A Synthetic Systems Programming Benchmark," *Comm. ACM*, Vol. 27, No. 10., Oct. 1984, pp. 1013-1030.
13. Y. Kashiwagi et al., "An Optimizing C Compiler for the Gmicro/500 Microprocessor," *Proc. Ninth Int'l Symp. TRON Project*, 1992, pp. 63-69.


Uchiyama


Arakawa


Narita


Aoki


Kawasaki


Matsui


Yamamota


Nakagawa


Kudo

Kunio Uchiyama is a senior researcher in Hitachi's Central Research Laboratory, where he holds responsibility for research and development of microprocessors. His interests include microprocessors, cache memories, and design automation tools.

Uchiyama received the BS and MS degrees in information science from the Tokyo Institute of Technology. He is a member of the IEEE.

Fumio Arakawa is a researcher in Hitachi's Central Research Laboratory, where he has been involved in 32-bit microprocessor design.

He received the BS and MS degrees in applied physics from the University of Tokyo.

Susumu Narita is a researcher in the Central Research Laboratory, where he has been involved with 32-bit microprocessor and design tools.

Narita received the BE and ME degrees in electrical engineering from Hokkaido University.

Hirokazu Aoki is a researcher in Hitachi's Central Research Laboratory, where he has been involved with cache memories and 32-bit microprocessor. He graduated from Shizuoka Technical High School.

Ikuya Kawasaki is an engineer in the Semiconductor and Integrated Circuits Division at Hitachi where he has been involved with 32-bit microprocessor. He holds responsibility for development of Gmicro/500.

He received the BS degree in electrical engineering from the University of Tokyo and the MS degree from the University of Washington.

Shigezumi Matsui is an engineer in the Semiconductor and Integrated Circuits Division at Hitachi, where he has been involved in 32-bit microprocessor design.

Matsui received the BE degree in mathematical engineering and instrumentation physics from the University of Tokyo.

Mitsuyoshi Yamamoto is an engineer at Hitachi's Semiconductor and Integrated Circuits Division, where he has been involved with 32-bit microprocessor.

He received the BE and ME degrees in electrical engineering from Nagoya University.

Norio Nakagawa is an engineer at Hitachi's Semiconductor and Integrated Circuits Division, where he has been involved with the 32-bit microprocessor and floating-point coprocessor. Nakagawa received the BS degree in physics from the Tokyo Institute of Technology.

Ikuo Kudo is an engineer in the Semiconductor and Integrated Circuits Division at Hitachi, where he has been involved with microprocessors for watch and automobile applications and with 32-bit microprocessors. Kudo graduated from Kitami Technical High School.

Address questions concerning this article to Kunio Uchiyama at Hitachi, Ltd., Central Research Laboratory, 1-280, Higashi-koigakubo, Kokubunji-shi, Tokyo 185, Japan; uchiyama@crl.hitachi.co.jp.

IEEE COMPUTER SOCIETY PRESS BOOKS

VLSI ALGORITHMS AND ARCHITECTURES Fundamentals

edited by N. Ranganathan

Addresses introductory and fundamental ideas related to VLSI algorithms and architectures and provides a concise tutorial on the topics. The chapters in this volume introduce basic concepts, present papers on systolic and wavefront arrays, focus on VLSI implementation of data structures, deal with VLSI structures for matrix and algebraic computations, and address important application areas. Overall, the book provides a variety of useful reference material for researchers and students seeking information on the design of hardware algorithms and architectures for VLSI implementation.

Sections: An Overview of VLSI Algorithms and Architectures, Systolic and Wavefront Arrays, Data Structures and Sorting, Matrix and Algebraic Computations, Pattern Matching and Text Retrieval, VLSI Processor Designs.

304 pages. August 1993. Hardcover. ISBN 0-8186-4392-7.
Catalog # 4392-01 — $40.00 Members $32.00

VLSI ALGORITHMS AND ARCHITECTURES Advanced Concepts

edited by N. Ranganathan

Contains many new studies exposing various computationally intensive problems requiring VLSI solutions and addressing advanced concepts as well as VLSI architectures for a broad spectrum of application areas. The chapters discuss important architectural design issues, focus on advanced concepts for systolic arrays and algorithms, and describe special-purpose architectures for a wide range of computationally intensive problems. The papers throughout this volume cover problems from a wide range of application areas that require VLSI solutions.

Sections: VLSI Architecture Design Issues; Advanced Topics in Systolic Arrays; Image, Speech, and Signal Processing; Artificial Intelligence and Computer Vision; Dictionary Machines and Data Compression; Iterative Algorithms.

312 pages. August 1993. Hardcover. ISBN 0-8186-4402-8.
Catalog # 4402-01 — $40.00 Members $32.00

CODES FOR DETECTING AND CORRECTING UNIDIRECTIONAL ERRORS

edited by Mario Blaum

This collection presents state-of-the-art theory and practice for codes that correct or detect unidirectional errors not covered by most books on error-correcting codes. It features key papers demonstrating the best results in each subject related to unidirectional errors. It is intended for engineers working in computer memory technologies and VLSI design, researchers in the area of error control codes, and graduate students needing access to this cutting-edge material.

Sections: Unidirectional Errors, Codes for Detecting Unidirectional Errors, Codes for Correcting Unidirectional Errors, Codes for Correcting *t*-Symmetric Errors and Detecting All Unidirectional Errors, Codes for Correcting and Detecting Combinations of Symmetric and Unidirectional Errors, Codes for Detecting and/or Correcting Unidirectional Burst Errors and Byte Errors.

224 pages. June 1993. Hardcover. ISBN 0-8186-4182-7.
Catalog # 4182-03 — $44.00 Members $35.00

REAL-TIME SYSTEMS Abstractions, Languages, and Design Methodologies

edited by Krishna M. Kavi

Presents important information on new formalisms, high-level programming languages, and CASE tools that increase the effectiveness of real-time systems. It includes survey articles, specifically written for this tutorial, and a collection of major research papers addressing formalisms, languages, and design methodologies. Its chapters describe some common myths regarding the use of formalisms in software development; detail the use of real-time temporal logic, process algebras, petri nets, and assertational methods; and examine formalisms based on operational semantics, axiomatic logic, and denotational semantics.

Sections: Real-Time Systems: Perspectives, Real-Time Specification and Verification, Real-Time Languages, Design Methodologies for Real-Time Systems.

672 pages. 1992. Hardcover. ISBN 0-8186-3152-X.
Catalog # 3152-01 — $70.00 Members $55.00

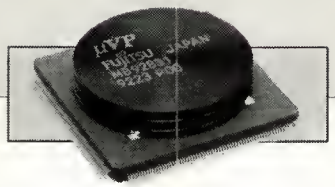
ALSO AVAILABLE !

VLSI DESIGN 1993
392 pages. January 1993. ISBN 0-1886-3180-5.
Catalog # 3180-02 – $80.00 Member $40.00

11TH ANNUAL 1993 VLSI TEST SYMPOSIUM
384 pages. April 1993. ISBN 0-1886-3830-3.
Catalog # 3830-02 – $70.00 Member $35.00

IEEE COMPUTER SOCIETY
10662 Los Vaqueros Circle
Los Alamitos, CA 90720

Call toll-free 1-800-CS-BOOKS
in CA (714) 821-8380
or FAX (714) 821-4641



The μVP 64-Bit Vector Coprocessor:

A New Implementation of High-Performance Numerical Computation

Intended for high-speed numerical computations, this coprocessor represents a true vector architecture on a single CMOS chip. It integrates 1.5 million transistors and offers 206-Mflops performance.

Makoto Awaga

Fujitsu Limited

Hiromasa Takahashi

Fujitsu Laboratories Limited

Although supercomputers are used extensively in scientific applications such as fluid dynamics and structural analysis, they are costly to purchase, operate, and maintain. Users seek less costly alternatives that make use of open systems for scientific applications to increase engineering workstation performance. Unfortunately, the superscalar, VLIW, or superpipeline architectures used in popular high-performance engineering workstations have been unable to match supercomputer performance. The μVP is a new engine for use in conjunction with those microprocessors; it offers the perfonmance of a supercomputer in open-system scientific applications.

The μVP is a single-chip vector coprocessor developed to meet the needs of high-performance processors in the coming years. It is the world's first supercomputer component implemented onto a single LSI (large-scale integrated) chip developed in CMOS technology. It introduces vector processing technology, a popular method for achieving high performance in the scientific computing community, to the microprocessor world. With 206-Mflops single-precision and 106-Mflops double-precision perfonmance at 50 MHz, the μVP offers a rate almost equivalent to the performance of typical mini-supercomputers.

Architecture

Most of the high-end scalar processors used in open systems are designed with pipelined architectures. Using new pipelined architectures (superpipeline, superscalar, VLIW) has significantly improved the performance of those scalar processors. However in some scientific applications in which a large amount of low locality data is accessed from a long running program, these scalar processors appear incapable of achieving their advertised performance. One of the main reasons is the physical limitation of the cache memory. In most of the scalar processors perfonmance depends on the cache hit ratio. Therefore, in applications having very low data locality but large volume, cache misses occur at every operand access, penalizing performance due to the need to refresh the cache contents.

We developed a function model of a superscalar processor and simulated this relationship between cache size and cache miss ratio. Our simulation used a model of DAXPY, which is a part of the Linpac benchmark program of Jack J. Dongarra at the University of Tennessee.[1] We took the Clean-up loop out of the source and put it in the two-dimensional loop shown in Figure 1.

N is assumed to be a multiple of 4. We checked the cache miss ratio, changing the matrix size of

0272-1732/93/1000-0024$03.00 © 1993 IEEE

the source data (value of *N*). Figure 2 indicates the result of our simulation. We configured the superscalar processor to have the following:

1) a 66-MHz clock frequency (a 15-ns cycle);
2) a capability for issuing up to three instructions every cycle: INT+INT+FP or INT+FP+FP (INT indicates integer operations and FP indicates floating-point operations);
3) a load/store operation between the data cache and registers of one operand each cycle;
4) four-cycle latencies for FPMUL and FPADD, and two-cycle latency for FPLOAD;
5) separate instruction and data caches (we configured the data cache as either 16-Kbyte entries, direct mapped, and a 32-byte line size or 32-Kbyte entries, two-way associativity, and a 32-byte line size);
6) a 10-cycle (150-ns) cache miss penalty; and
7) a 64-entry, 32 sets × two-way set associative, 4-Kbyte-page data TLB (translation look-aside buffer) with a 30-cycle (450-ns) miss penalty.

We converted the source program based on the DAXPY Clean-up loop into the codes in our simulation according to the specified configuration in Figure 3.

Although our simulation achieved a peak performance of 132 Mflops at 66 MHz (two floating-point operations can be executed at a time), once the matrix size exceeds 100×100 the 10 percent cache miss causes a significant decrease in sustained performance. We attempted four iterations of loop unrolling to minimize the overhead of conditional branches. However, the sustained performance remained at less than 20 percent of peak when the matrix size was greater than 100×100. This simulation assumed a TLB hit of 100 percent and therefore did not include TLB penalties.

A vector register machine performs all vector operations except loads and stores on data stored in vector registers. Unlike cache memory architectures, software con-

```
DO 20 I = 1,N
   DO 10 J = 1,N,4                                  ------
      DY(I, J) = DY(I, J)+DA*DX(I, J)               |
      DY(I, J+1) = DY(I, J+1)+DA*DX(I, J+1)         |   CLEAN-UP LOOP
      DY(I, J+2) = DY(I, J+2)+DA*DX(I, J+2)         |   part of DAXPY
      DY(I, J+3) = DY(I, J+3)+DA*DX(I, J+3)         |
   10 CONTINUE                                      ------
20 CONTINUE
```

Figure 1. Two-dimensional loop.

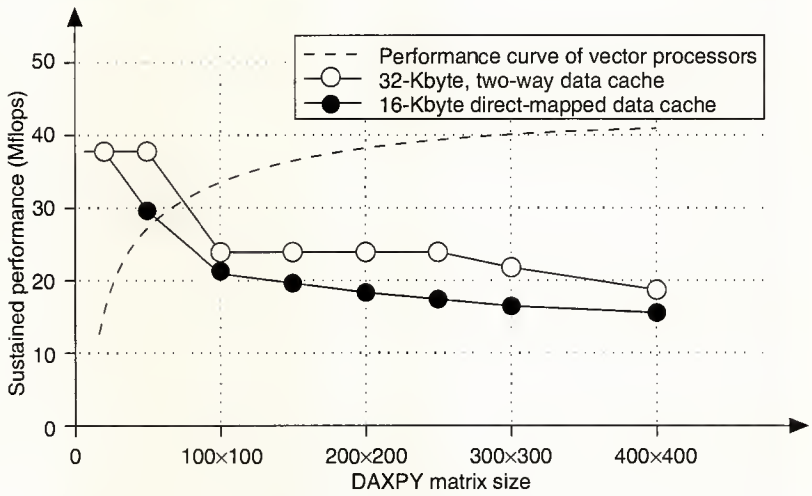


Figure 2. Simulation results of a superscalar processor's sustained performance.

	Load/Store	FP mul/add	Integer operation	
Loop:	ld.d Y[i+3]	fmul.d Z[i]	add p X[i]	(1)
	ld.d X[i]	fmul.d Z[i+1]	add p X[i+1]	(2)
	ld.d X[i+1]	fmul.d Z[i+2]	add p X[i+2]	(3)
	ld.d X[i+2]	fmul.d Z[i+3]	add p X[i+3]	(4)
	ld.d X[i+3]	fadd.d Z[i]	add p Z[i]	(5)
	st.d Z[i]	fadd.d Z[i+1]	add p Z[i+1]	(6)
	st.d Z[i+1]	fadd.d Z[i+2]	add p Z[i+2]	(7)
	st.d Z[i+2]	fadd.d Z[i+3]	add p Z[i+3]	(8)
	st.d Z[i+3]		add p Y[i]	(9)
	ld.d Y[i]*		add p Y[i+1]	(10)
	ld.d Y[i+1]*		add p Y[i+2]	(11)
		sub cc	add p Y[i+3]	(12)
	ld.d Y[i+2]*		brc loop	(13)

Figure 3. Converted codes for simulation. (* indicates software pipelining used.)

Table 1. Product lineup.

Item	Description
Part numbers	MB92831-33/-45/-50
Operation frequency	33/45/50 MHz
External bus configuration	
Data bus	64 bits
Address bus	32 bits
Bus bandwidth	264/360/400 Mbytes/s
Internal registers	
Vector (VR)	8 Kbytes
Mask (VMR)	64 bytes
Scalar (VSR)	128 bytes
Command buffer (VCB)	1 Kbyte
Address translation (TR)	512 bytes
Control	11
Internal execution pipelines	
Add	64-bit Adderx1 or 32-bit Adderx2
MUL	64-bit Muliplierx1 or 32-bit Multiplierx2
DIV	64-bit Dividerx1
Graph	Graphic operation
Mask	Mask operation
Applicable data format	Single-/double-precision floating-point (IEEE Std. 745) 32-bit integer/logical/pixel
Command format	32-bit, fixed
No. of commands/type	141 vector operation 57 scalar operation 9 general control
Peak performance	
Single-precision	136/186/206 Mflops
Double-precision	70/96/106 Mflops
Debugging support	IEEE Std. 1149.1, JTAG
Supply voltage	3.3V
Power dissipation	3.0/4.5/5.0 watts
Package	Ceramic SQFP-256 (0.5-mm lead pitch)

trols the contents of vector registers, thus avoiding performance constrictions caused by cache misses. In comparison, vector processors tend to exhibit performance curves as shown by the dashed line in Figure 2. Due to the start-up overhead of vector processors, superscalar processors achieve better performance when the matrix size is less than 50×50. However, once the matrix size exceeds 50×50, vector processor performance increases, and the increase is maintained regardless of the matrix size.

Since we designed the μVP based on this vector register machine architecture, it provides several important properties of vector operations:[2]

- no overhead for branch condition control,
- low cost of latency to main memory access, and
- vector instructions that replace entire loops.

Design considerations

In the scientific computation arena, where very large active data sets are accessed with low locality, vector supercomputers enjoy great popularity due to the useful properties just discussed. However, many users normally share these vector supercomputers, and a huge amount of initial investment and running costs is required.

To take advantage of vector operations in personal systems and drastically improve the numerical computation performance, we implemented a vector architecture onto a single CMOS LSI chip. In this implementation we tried to

- reduce the start-up overhead caused by the pipeline latency,
- make a host processor interface universal, so that it can be attached and used with various types of microprocessors,
- build a memory interface using conventional DRAMs and offer enough speed to synchronize to internal operations,
- put all the necessary execution pipelines and reasonable size of vector registers on a chip for which reasonable production yield can be expected, and
- mount the chip in a reasonably small package so it can be used for open-system products and/or cost sensitive board-level products.

Product overview

Table 1 lists the main features of the μVP, which integrates approximately 1.5 million transistors into a 15.99×15.99-mm CMOS chip. Figures 4 and 5 show a die photo and the packaged μVP. The μVP contains three separated execution pipelines: multiplier, adder/graphic, and divider/mask. While these three execution pipelines work in parallel, the coprocessor transfers an operand between internal register files and external memory through the load/store pipeline. The μVP achieves the peak performances listed in Table 1 when these three execution pipelines work simultaneously at 50-MHz operation.

Figure 4. Die photo.

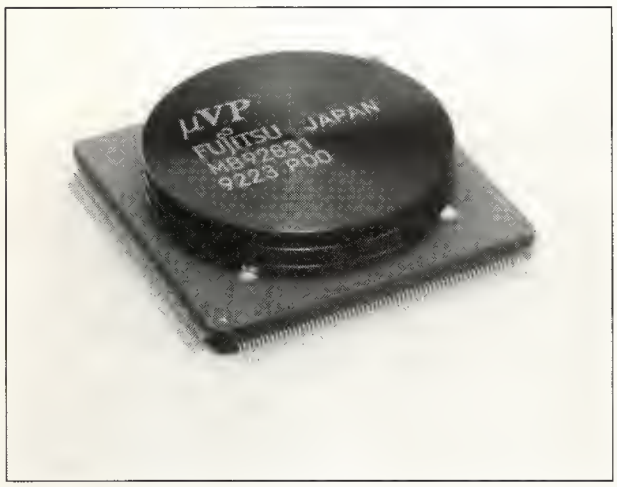


Figure 5. μVP package appearance.

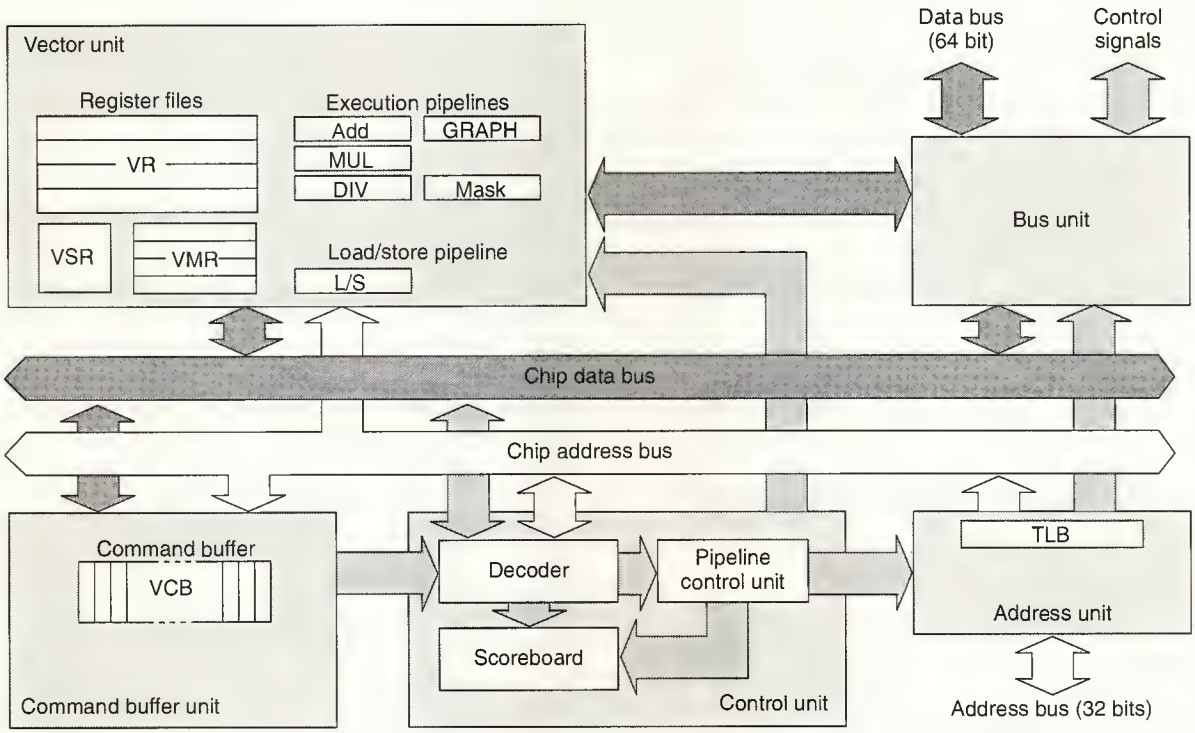


Figure 6. μVP block diagram.

Figure 6's block diagram shows that the μVP contains five units: vector, command buffer, control, address, and bus. Among them, the vector unit is the main engine that performs all the calculations (see Figure 7 for its block diagram). This unit consists of various execution pipelines, a load/store pipeline, and register files.

Execution pipelines. To handle numerical calculations, the μVP contains multiplier (MUL), adder (Add), divider (DIV),

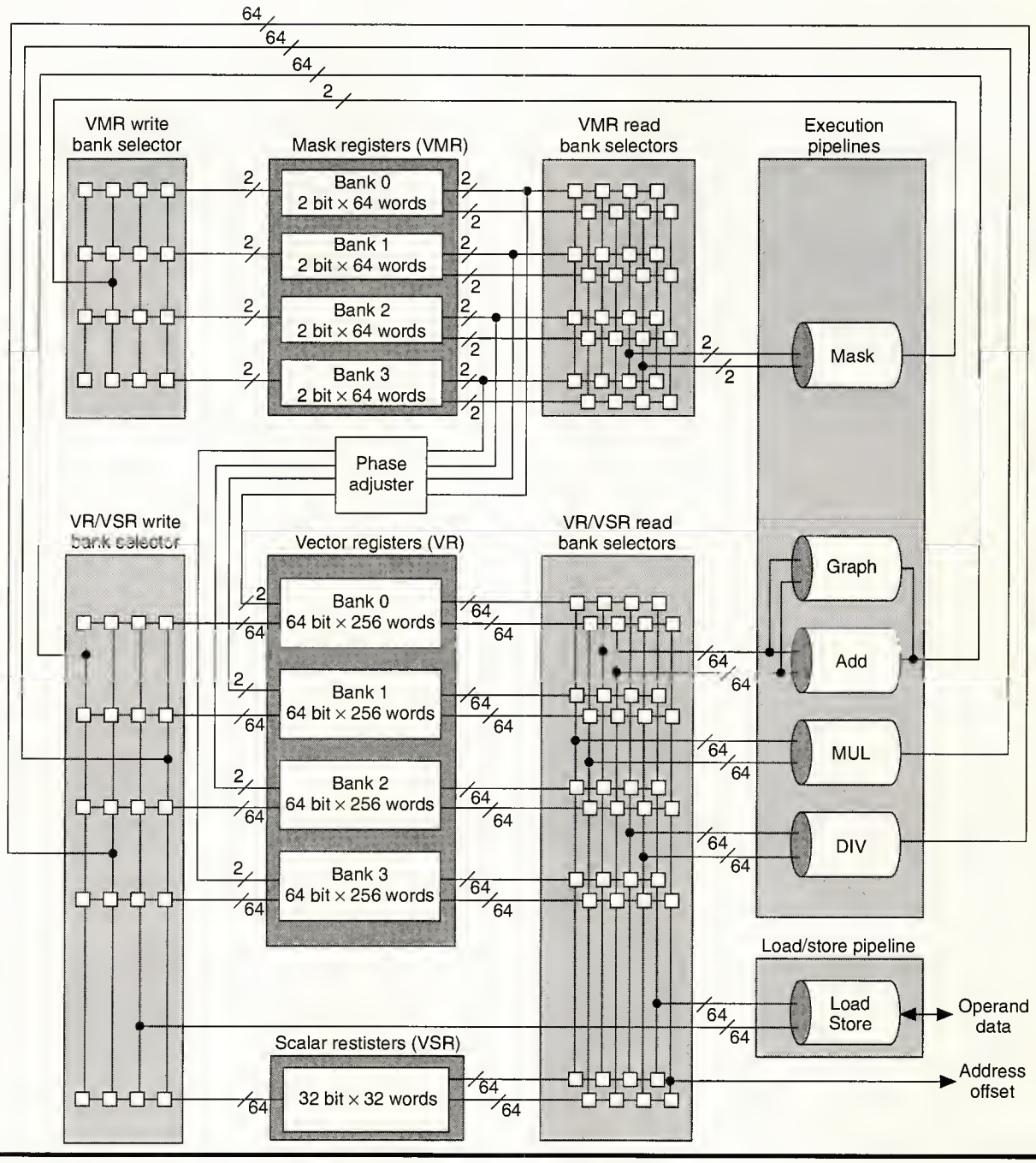


Figure 7. Vector unit block diagram.

graphic operation (Graph), and mask operation (Mask) pipelines. All five pipelines work independently. To work simultaneously, each pipeline has a pair of 64-bit-wide data loading paths and one 64-bit data storing path to register files. However, since Add and Graph share the same data paths, one of them is automatically selected according to the type of commands. Since DIV and Mask are assigned to the same start-up phase timing, these two do not work simultaneously either.

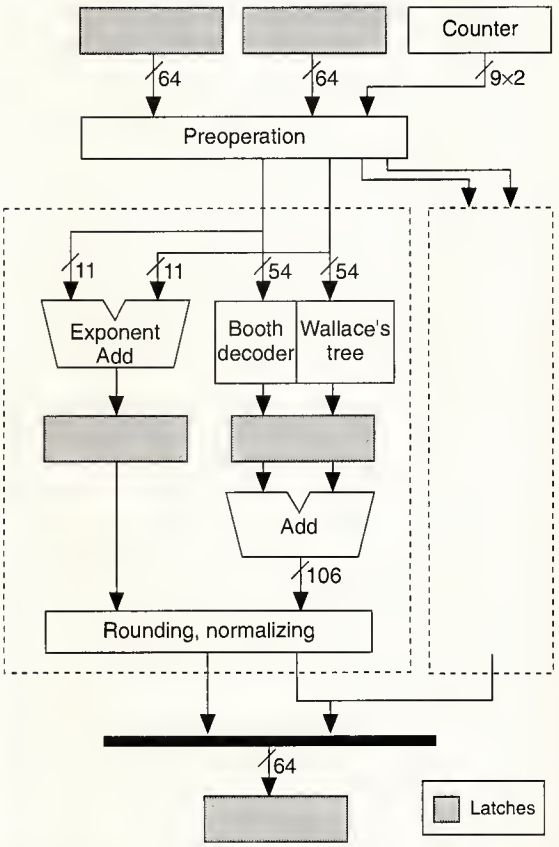


Figure 8. MUL block diagram.

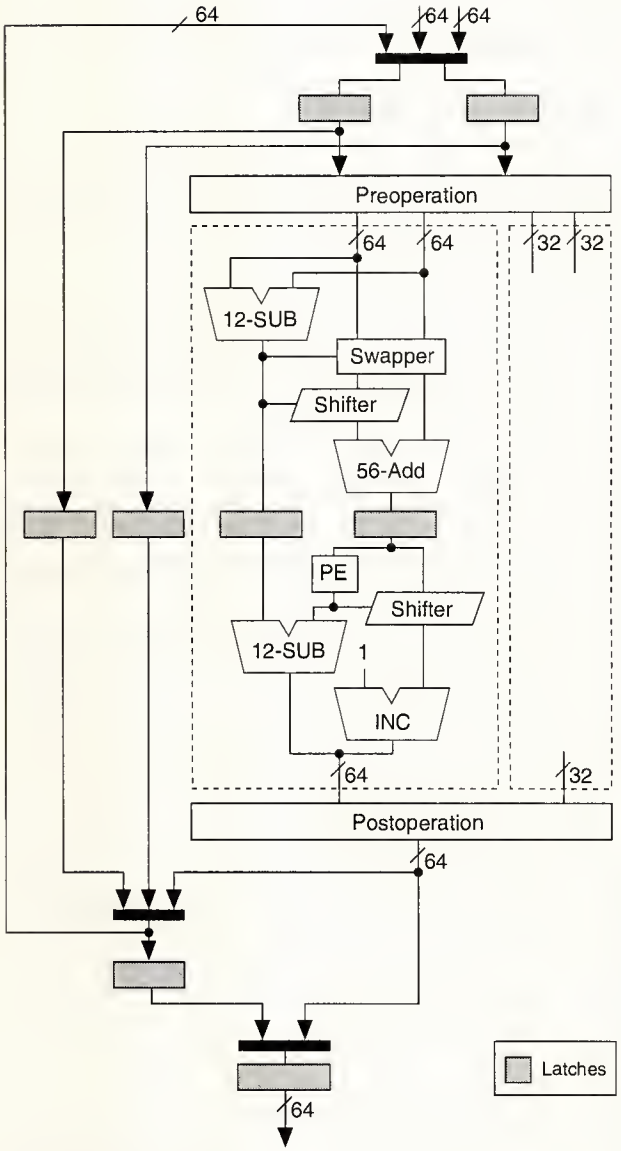


Figure 9. Add block diagram.

To minimize the start-up overhead, MUL, Add, Graph, and Mask are formed by four stages (source data reads, execution/first half, execution/second half, and destination store). One calculation takes four clock cycles. Since they all work in a pipelined manner, at every cycle a new pair of source data is fed into a pipeline, and at the same time a new destination datum is stored back to the register file. So one calculation completes each cycle. Figures 8 and 9 show the block diagrams of MUL and Add. Both MUL and Add have a single-precision and a double-precision execution plane and a separate single-precision-only execution plane; two single-precision calculations execute at the same time. Therefore at 50 MHz, both of them perform at 50 Mflops for double-precision calculations and 100 Mflops for single-precision calculations. DIV consists of two divider planes, both implemented using iterative radix-16, nonrestoring division.[3] Each of those divider planes offers 1/16th the performance of MUL and Add, regardless of the data type; so DIV's total performance is 6.25 Mflops at 50 MHz.

When these pipelines work simultaneously, the μVP shows peak performance. When both MUL and Add execute single-precision calculations, while DIV works, this peak performance is an addition of those execution pipelines' performance, which is 206 Mflops at 50 MHz.

Load/store pipeline. The 64-bit-wide load/store pipeline (L/S) transfers data between internal register files and external memory. It simultaneously transfers one 64-bit datum or two 32-bit data every clock cycle to the internal pipeline executions.

Vector register. The vector register (VR) contains 8 Kbytes

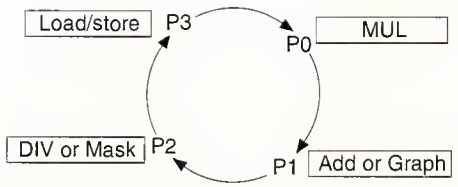


Figure 10. Scheduling of pipeline start timing.

of source and destination vector operands as well as the address offset information for indirect-addressing memory access. The VR consists of three-port SRAMs, so that two read operations and one write operation can take place at the same time. The entire four-way bank organization register file can schedule up to four pipelines to connect to those four VR banks, individually, each clock cycle. Thus, data transfers between VR and pipelines efficiently and simultaneously.

The VR is automatically divided into 8, 16, 32, or 64 partitions (VR0 to VR63), depending on the definition of vector length. The element unit of the VR is 64 bits long. One word of double-precision data or two words of single-precision data can be stored in each element unit. When the VR is divided into eight partitions making each partition 1 Kbyte (64 bits×128 words), each partition can hold 128 elements. This means that the maximum applicable vector length for the μVP is 128 in double-precision or 256 in single-precision calculations.

Before the design of the μVP started, we considered the expected process yield and decided that the chip size should be smaller than 16 mm×16 mm. Although the bigger VR size is always better for vector operations, we enforced this upper limit to the chip. When we developed the floor plan, we considered a rough proportion of the area to be occupied by all other logic parts and such factors of the VR as the cell size, organization, and reasonable capacity. We set the VR size at 8 Kbytes.

Scalar registers. The scalar registers (VSR0 to VSR31) are general-purpose data registers. They contain scalar operands and various parameters to be used in μVP operations. Typical operations involve a load/store start address, vector stride (a distance between vector elements located next to each other in memory), and so on. Like the VR, the VSR element is also a three-port SRAM. The scalar registers are directly mapped to a host CPU's address field, so that the host CPU can access any of these registers directly.

Mask register. The Mask register (VMR) contains 64 bytes of register file. To vectorize conditional branch operations, this register file stores mask information for the correlated vector operand. Like the VR, the VMR is formed in a four-way bank structure. The element of the VMR is also a three-port SRAM, and the entire register file is divided into 2, 4, 8, or 16 partitions, according to the definition of vector length.

The functional mechanism

To perform vector calculations in high speed simultaneously, we invented a state machine control mechanism called the Bank Slot Phase System.[4] This phase control system realizes completely synchronized parallel operations of various pipeline executions, data transfers between register files and execution pipelines, and data transfers between register files and external memory. All the operations handled by the μVP are driven by a Bank Slot Phase clock, which repeats four phases: P1, P2, P3, and P4. The start-up timing of each execution pipeline or load/store pipeline is assigned to one of those clock phases, as shown in Figure 10.

The Bank Slot Phase clock maintains strict control over all data transfer operations between each execution pipeline and the register files (VR, VSR, or VMR) to avoid access conflicts to the same VR bank from multiple pipelines. To control all these timings by hardware, we adopted data distribution management control circuits called bank selectors. Two sets of bank selectors control data loading (transfer from register files to pipelines), while one set controls data storing (transfers from pipelines to register files). Each port of bank selectors is 64-bits wide.

Read bank selectors transfer two sets of 64-bit-wide source data at every clock from register files to all three execution pipelines. The write bank selector transfers 64-bit-wide destination data from all execution pipelines to register files. At the same time the selectors transfer 64-bit-wide data between the load/store pipeline and the register files. Up to seven sets of 64-bit-wide data can transfer from register files to pipelines through read bank selectors, or four sets of 64-bit-wide data can transfer in the opposite way through the write bank selector at every clock cycle. Thus, the bandwidth of those bank selectors are 2,800 Mbytes/s for reads and 1,600 Mbytes/s for writes at 50 MHz.

These bank selectors smoothly switch control for the data transfer between register files and vector pipelines (execution pipelines and load/store pipeline). Table 2 lists the data path assignments at each clock phase controlled by these bank selectors.

Since the selectors are organized by shuffle circuits consisting of multiple tristate drivers (see Figure 11, on p. 32), the physical area of bank selectors is reduced to approximately one fourth of that formed by conventional AND-OR gate arrays.

The efficiency of simultaneous vector executions depends on such conditions as a register conflict or a pipeline hazard. (A conflict occurs when multiple pipelines access the same register at the same time. A hazard occurs when subsequent instructions attempt to use a pipeline already in use.) The program shown in Figure 12a on page 33, for example, does not have conditions that prevent execution pipelines from working in the most efficient way. So in this case, the bank selector executes all four instructions simultaneously, as shown

Table 2. Data transfer path assignment at each clock phase. Screened areas are undefined.

Register files		Bank selectors	Phase number of Bank Slot Phase Clock			
			P0	P1	P2	P3
Vector register (VR)	Bank 0	Read #0	MUL	Add/Graph	DIV*	Store
		Read #1				Address offset **
		Write	Add/	DIV*	Load	MUL
	Bank 1	Read #0	Store	MUL	Add/Graph	DIV *
		Read #1	Address offset**			
		Write	MUL	Add/	DIV*	Load
	Bank 2	Read #0	DIV*	Store	MUL	Add/Graph
		Read #1		Address offset**		
		Write	Load	MUL	Add/	DIV*
	Bank 3	Read #0	Add/Graph	DIV*	Store	MUL
		Read #1			Address offset **	
		Write	DIV*	Load	MUL	Add/
Mask register (VMR)	Bank 0	Read #0			Mask*	
		Read #1				
		Write		Mask*		
	Bank 1	Read #0				Mask*
		Read #1				
		Write			Mask*	
	Bank 2	Read #0	Mask*			
		Read #1				
		Write				Mask*
	Bank 3	Read #0		Mask*		
		Read #1				
		Write	Mask*			

* DIV and Mask do not work simultaneously.
** Applied when indirect addressing mode in use.

in the time sequence chart of Figure 13a on p. 33.

However, in case of a program as shown in Figure 12b, the destination register of the prior vector multiply instruction is referred to as a source operand by the following vector add instruction, and a register conflict occurs between these two instructions. In this case until at least one element is fed back to VR8 by the VMULD instruction, the following VADDD instruction will not be started. But once the products are started after being written back to VR8, all the subsequent operations of VADDD and VMULD can execute simultaneously. Figure 13b shows a time sequence of this operation.

In case of a program similar to the one shown in Figure 12c, instructions to use the same MUL pipeline are put next to each other. Since there is only one MUL in the vector unit, two multiply instructions cannot execute at the same time. Therefore, the following VMULD instruction (2) is not started until the prior VMULD instruction (1) completes. Figure 13c shows a time sequence of this operation.

These conflict conditions are automatically detected by a scoreboard unit residing in the control unit. Once a conflict is detected, the coprocessor automatically postpones execution of the following instruction, which caused the conflict, until that condition clears. Programmers don't need to worry about the overrun and/or pipeline scheduling. Also, as shown in Figure 13b, once the conflict condition clears, all the subsequent operations proceed simultaneously. So the longer the vector length, the less the conflict overhead becomes relative.

Bus configuration

To make handling of the package easy, we tried to keep the package as small as possible. The most useful way to do so is to minimize either pin count or lead pitch. Since the μVP is a 64-bit vector coprocessor to be used with a 32-bit microprocessor, the data bus and address bus need to be 64-bits wide and 32-bits wide. This means that one set of the load/store

continued on p. 34

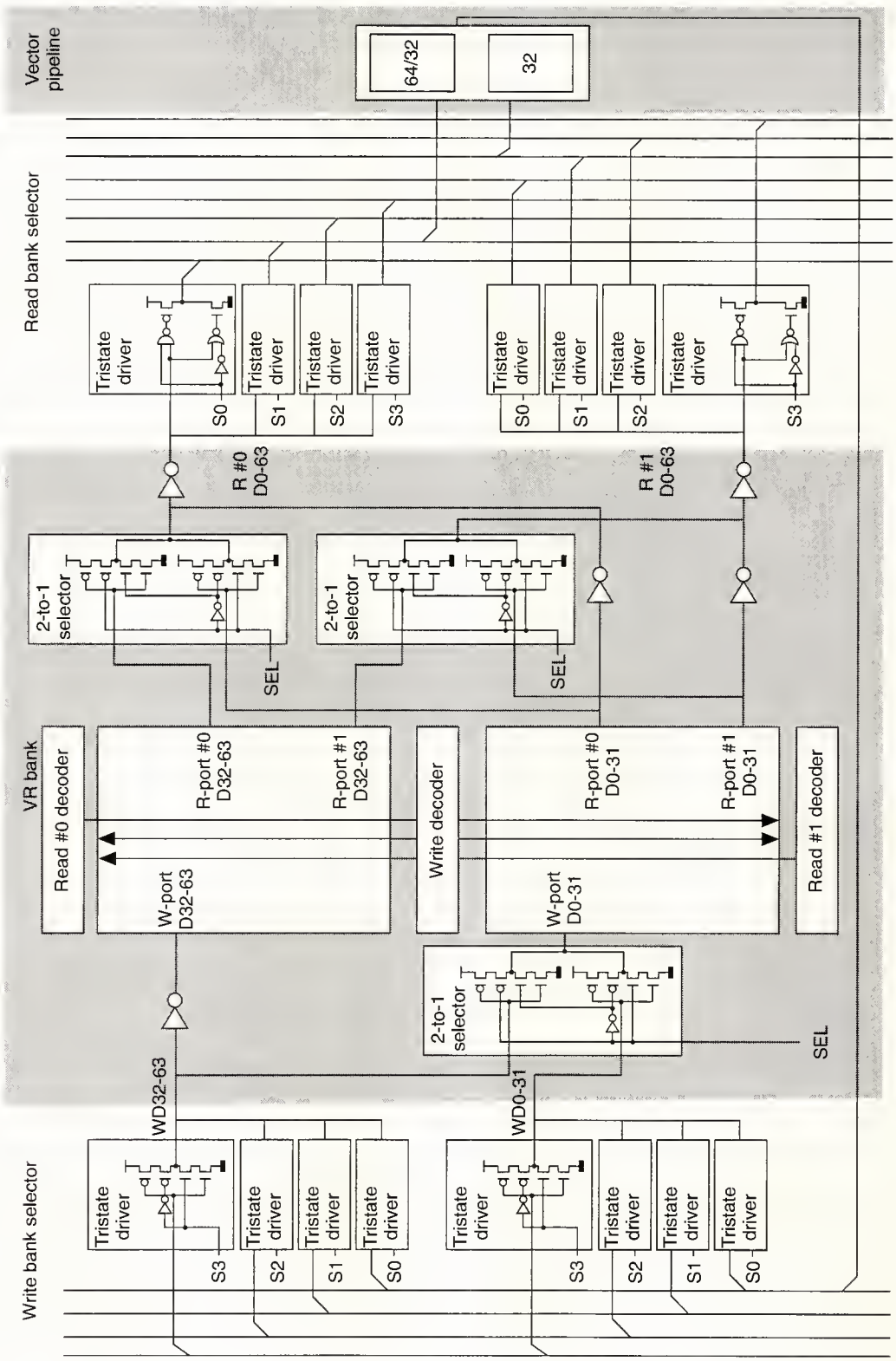


Figure 11. Bank selector circuits.

```
                                ; n : vector length (n≤128 in D.P. floating point operand)
                                ;
vmuld     vr0, vr4, vr8         ; D.P. vector multiply      VR0(1:n)*VR4(1:n)=VR8(1:n)
vaddd     vr12, vr16, vr20      ; D.P. vector add           VR12(1:n)+VR16(1:n)=VR20(1:n)
vdivd     vr24, vr28, vr30      ; D.P. vector divide        VR28(1:n)/VR24(1:n)=VR30(1:n)
vst64     vcr36, vsr0, vsr1     ; D.P. vector operand store
(a)

vmuld     vr0, vr4, vr8         ; (1) D.P. vector multiply
vaddd     vr8, vr12, vr16       ; (2) D.P. vector add       [start when VR8(1) (first element of above (1)
                                                               vector multiply) will be ready]
vst64     vr16, vsr0, vsr1      ; D.P. vector operand store [start when VR16(1) (first element of above (2)
(b)                                                            vector add) will be ready]

vmuld     vr0, vr4, vr8         ; (1) D.P. vector multiply
vmuld     vr8, vr12, vr16       ; (2) D.P. vector multiply  [start when above (1) vector multiple will all
                                                               complete]
vst64     vr16, vsr0, vsr1      ; D.P. vector operand store [start when VR16(1) (first element of above (2)
(c)                                                            vector multiply) will be ready]
```

Figure 12. Programming models: to get the best performance (a), cause a register conflict (b), and cause a pipeline hazard (c).

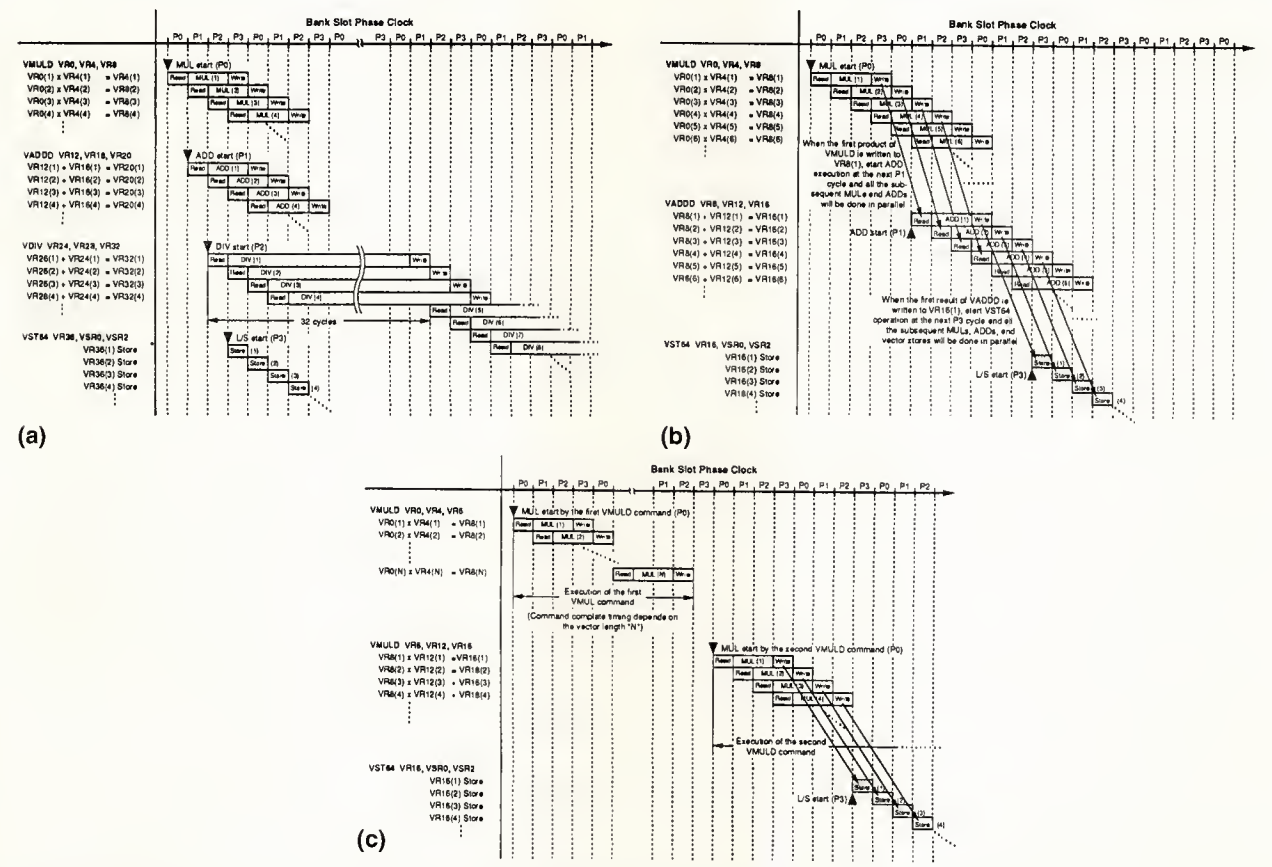


Figure 13. Time sequence charts: maximum efficiency (a), register conflict (b), and pipeline hazard (c).

```
C*
C*     COORDINATE TRANSFORMATION
C*           (OBJECT —> WORLD)
C*
C*     SOURCE        : X1(N)=SP(1,N), Y1(N)=
C*                     SP(2,N), Z1(N)=SP(3,N)
C*     DESTINATION   : X2(N)=DP(1,N), Y2(N)=
C*                     DP(2,N), Z2(N)=DP(3,N)
C*
C*************************************************
C
      INTEGER*      4 N
      REAL*4        SP(3,N),DP(3,N)
      REAL*4        CD(4)
C
      DO 120 I=1,N
       DO 110 J=1,3
        CD(J)=0.0
        DO 100 K=1,3
         CD(J)=CD(J)+SP(K,N)*CX(K, J)
100      CONTINUE
        CD(J)=CD(J)+CX(4, J)
110      CONTINUE
       DP(1,N)=CD(1)
       DP(2,N)=CD(2)
       DP(3,N)=CD(3)
120    CONTINUE
```

Figure 14. Coordinate transformation Fortran source program.

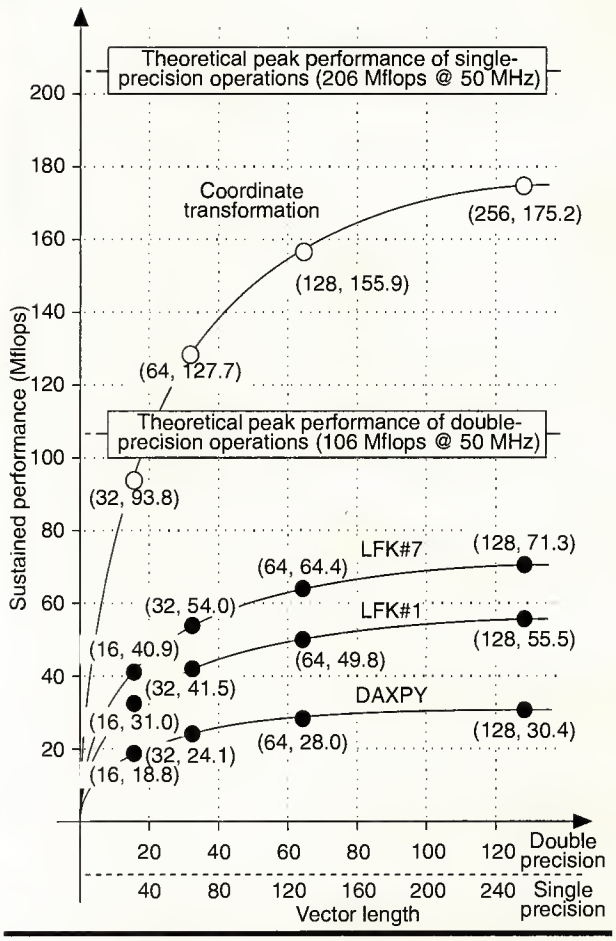


Figure 15. μVP performance simulation results. Open dots indicate single-precision operations and closed dots, double-precision.

pipeline requires almost 100 pins to communicate with external memories. Considering this point, we equipped just one load/store pipeline, although there are three parallel execution pipelines (MUL, Add/Graph, and DIV/Mask). We kept the total external signal pin count within 130. The chip is mounted in a 1.5×1.5-in., 256 SQFP (shrink quad flat pack), which made it possible to use the μVP in such cost-sensitive applications as add-on accelerator board products.

A universal host interface makes it possible to use the μVP with various types of host CPUs. All the internal registers except VR and VMR are address mapped in the host CPU's address space. By issuing the appropriate address and data, the host CPU can access any of these registers while the μVP is in the idle state.

Vector instructions that access memory have a known access pattern. If the vector's elements are all adjacent, fetching the vector from a set of heavily interleaved memory banks works well.[5] However, if the memory consists of too many banks, we cannot achieve good performance in short vector length operations due to a high memory access latency. Since the μVP is a CMOS single chip, we cannot organize as huge an external memory system as that used in large supercomputers.

As a result, we decided to use a four-way interleaved memory. With consecutive element order, the memory access latency is four cycles. When the μVP works at 50 MHz and each element access can be completed within 80 ns, no wait state occurs and all the vector operations take place fully synchronized. Even if the interleaved memory is organized by relatively slow DRAMs, this access speed is possible in high-speed page mode.

Command set

The μVP supports 207 commands. They include 141 vector operation commands, 57 scalar operation commands, and 9 general control commands such as branches and register-to-register internal data transfers. All commands are 32-bits

long and fixed; they contain 10 bits of opcode field and 22 bits of register direction field. We studied our company's command set for vector supercomputers and tried to implement similar μVP vector operation commands that are optimized to scientific calculations. Those vector operation commands support such complex operations as max/min value find and vector sum to improve the efficiency of vectorization. Scalar operation commands are separately supported from vector operation commands, though scalar operation commands are functionally identical to the vector operation commands with the vector length setting of 1. This approach executes scalar operations simultaneously while vector operations are taking place, without changing the vector length setting.

Program coding example

Figure 14, contains a coding sample of a Fortran source program for coordinate transformation. The μVP applicable vector length is up to 256 in single-precision calculations. If the element number in the source program is bigger than 256, all data elements must be divided into numbers of segments that the μVP can manage in bulk vector operation. For each segment, the same set of vector operation is repeated. Although repetition control is required, this overhead is small enough to ignore in comparison to the volume of the vector operation.

Performance evaluation

We simulated the coordinate transformation program sample in our μVP model. We also used the DAXPY Clean-up loop (a single loop) in the cache miss ratio simulation of a superscalar processor. Other than these two programs, we picked up No. 1 and No. 7 loops from the Livermore Fortran Kernel (LFK), which is a test suite produced by Lawrence Livermore National Laboratory. These three additional simulated benchmark loops check sustained performance in double-precision floating-point operations.

Figure 15 shows the simulation results of these loops. In this figure the three curves marked with dark dots show the results of the double-precision operation of LFK#7, LFK#1, and DAXPY. The curve marked with white dots shows the result of the single-precision operation of the coordinate transformation program. Sustained single-precision performance of this coordinate transformation program is 175.2 Mflops at 50 MHz. The sustained performance is about 80 percent of the theoretical peak performance of 206 Mflops. Since the proportion of internal MUL and add operations is relatively high toward the load/store operations in this coordinate transformation, all the pipeline operations are well synchronized, and the sustained performance comes close to the theoretical peak.

We achieved a sustained performance of 30.4 Mflops, 55.5 Mflops, and 71.3 Mflops at 50 MHz for the DAXPY, LFK#1, and LFK#7 benchmarks. In DAXPY the vector load/store command appears three times as many as the vector MUL or vector add command. Since the μVP has only one load/store pipeline, these three load/store commands do not execute in parallel. Vector MUL and vector add executions overlap with one load command execution. But while the other load command and store command execute, all the internal execution pipelines stay in the idle state. This accounts for the sustained DAXPY performance being about 30 percent of the peak performance (106 Mflops at 50 MHz).

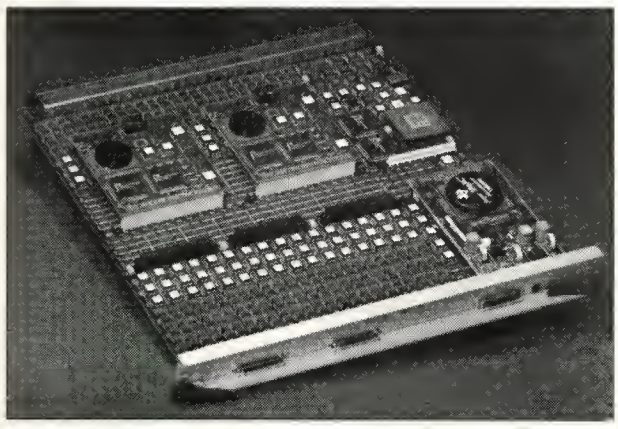


Figure 16. VPE application board.

As the result of all these simulations, we confirmed that the sustained performance curves of the μVP follow the characteristics of vector operation shown in Figure 1. As the vector length becomes longer, the sustained performance reaches a certain peak level. There is no such performance drop, as shown in the simulation of a superscalar processor caused by a cache miss. Also, from these results we confirmed that the μVP provides a quick upward performance curve because of the short pipeline latency. At vector length 16, the sustained perfonmance already reaches 50 percent of the peak of each curve.

We also measured the sustained performance of a real system using the μVP. This measurement was done on a VPE (vector processor element) board for a modular, massively parallel processing system called CS-2 developed by Meiko.[6] This system was introduced at Supercomputing 1992. Lawrence Livermore National Laboratory plans to install a 256-vector processor element version of this system this year.

Figure 16 shows this VPE board with a couple of μVPs placed under the control of a SuperSparc chip. Those three processors share a memory system with 128 Mbytes of external DRAM in 16 independent banks. We measured this performance at 33 MHz of operation frequency for three benchmark test programs: DAXPY, DDOT, and DGEMM (matrix multiply). Table 3, next page, lists the results of this evaluation.

Two μVPs work simultaneously on this board, so the performance result appears to be almost double the single-chip

Table 3. VPE simulation results (Mflops).

Benchmark	At 45 MHz	At 50 MHz
DAXPY (N = 8,192)	53.7	59.6
DDOT (N = 8,192)	70.6	78.4
DGEMM (N = 1,000×1,000)	129.3	143.5

simulation result, according to the DAXPY result comparison. But because of such overhead factors as μVP start-up from SuperSparc, periodic DRAM refresh cycles, internal repetition control of the μVP program, and so on, the real result per chip is a little less than the simulation result.

Other than Meiko's CS-2, various kinds of application systems of the μVP are now available or under development. Currently, the most popular applications of the μVP are standard bus adapters like a VME board and add-on accelerator cards for PCs and workstations.

THE VECTOR ARCHITECTURE IS WELL KNOWN in the supercomputer world as a method of achieving high performance for floating-point operations. As open systems become more and more popular in this widely expanding field however, a growing number of users require additional computation power. To meet this trend, designers have significantly improved the processor power of those systems by introducing new architectures. However, applications, such as aerodynamic simulation, structural analysis, and seismic processing, require even higher levels of performance. We consider the μVP to be an alternative to the high-speed floating-point computation for such open systems as engineering workstations and back-end computing servers. With the μVP those applications can be executed in a more personal environment.

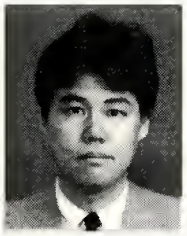
The design background and implementation of the μVP required the investment of various hardware facilities and implementation techniques of vector architecture so it could fit on a single CMOS LSI chip. The μVP has been in production as MB92831-33/45/50 since last November, and volume production parts are available now.

Acknowledgments

We thank all μVP project members and parties who were involved with the chip and support environment development. We also thank Richard Cownie of Meiko, who helped us provide performance evaluation information of a real chip.

References

1. J. Dongarra et al., *LINPACK User's Guide*, SIAM, Philadelphia, Penn., 1979.
2. P.M. Kogge, *The Architecture of Pipelined Computers*, Hemisphere Publishing Corporation, 1981.
3. K. Hwang, *Computer Arithmetic Principles, Architecture, and Design*, John Wiley & Sons, New York, 1979.
4. H. Iino et al., "A 289-Mflops Single-Chip Supercomputer," *ISSCC Digest Tech. Papers*, Feb. 1992, pp. 112-113.
5. J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., 1990.
6. *CS-2 Product Description*, Meiko, 1992.



Makoto Awaga, a manager of the μVP engineering team at Fujitsu's Kawasaki site, has been involved with strategic marketing and product planning of high-end microprocessors and controllers. He has worked on the company's personal computer development and in San Jose, California, on microcomputer marketing and research. Awaga received a BS degree in electrical engineering from Ritsumeikan University, Kyoto.

Hiromasa Takahashi is a researching manager in the Processor Laboratory at Fujitsu Laboratories Limited in Atugi, Japan. His research interests include microprocessors and architecture for high-speed processing systems.

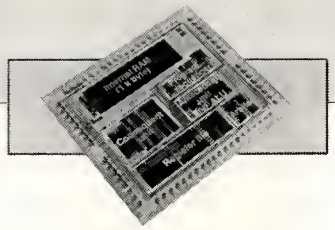
Takahashi received BE and ME degrees in electrical engineering from Chiba University. He is a member of the Institute of Electronics, Information and Communication Engineers of Japan.

Direct any questions concerning this article to Makoto Awaga, Fujitsu Limited, 1015, Kamikodanaka, Nakahara-ku, Kawasaki 211, Japan; or awaga@gmd.ed.fujitsu.co.jp.; or Hiromasa Takahashi, Fujitsu Laboratories Limited, 10-1, Morinosato-Wakamiya, Atsugi 243-01, Japan; or taka@prd.cs.fujitsu.co.jp.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate numbers on the Reader Service Card.

Low 159 Medium 160 High 161



Fuzzy Inference and Fuzzy Inference Processor

Fuzzy inference, a data processing method based on the fuzzy theory, has found wide use, mainly in the control field. Consumer electronics, which accounts for most current applications of this concept, does not require very high speeds. Though software running on a conventional microprocessor can perform these inferences, high-speed control applications require much greater speeds. We have devised a processor that operates at 200,000 fuzzy logic inferences per second. Our design features 12-bit input and 16-bit output resolution.

Kazuo Nakamura

Narumi Sakashita

Yasuhiko Nitta

Ken'ichi Shimomura

Takeshi Tokuda

Mitsubishi Electric Corporation

As originally proposed by Zadeh,[1] the fuzzy theory treats ambiguous concepts as mathematical numbers or functions. Fuzzy inference, a data processing method based on this theory, has found widespread use, mainly in controllers, and numerous reports attest to its success.[2] See the Fuzzy theory box on page 39 for further background on this novel and far-reaching concept.

Figure 1 (next page) compares the performance of several implementations of the fuzzy inference. The graph's horizontal axis represents the inference speed measured in units of fuzzy logic interferences per second, or FLIPS. Its vertical axis represents the input resolution. A software approach using a conventional microprocessor can attain up to 1 KFLIPS with 8 to 16 bits of resolution, sufficient for most current consumer electronics applications.

High-speed control applications, however, such as automobile engine control, demand greater inference speed. These applications will require dedicated, large-scale integration that partially or fully perfoms the inference in hardware. Although several chips have been introduced,[3] those operating at higher speeds cannot attain high resolution. Applications demanding sophisticated control will require high resolution as well as high speed. Here we describe both the fuzzy inference mechanism and the high-speed, high-resolution fuzzy inference processor we have developed to meet these needs.

Fuzzy inference

To understand the fuzzy inference mechanism, we need to take a look at several relevant concepts. We use the example of a representative control system to help us along.

Control system. Figure 2 shows a simple control system. In the figure, "state variable" represents the state of the controlled object. If we take an air conditioner as a controller, for example, the temperature of the room is the state variable. "Reference" is the target value of the state variable—the temperature of a cooler setting. "Disturbance" represents the uncontrollable outside force that changes the state variable—the temperature outdoors.

The control system needs to keep the state variable as closely equal to the reference as possible, or to change the state variable to the reference as quickly as possible. For this purpose, the controller calculates the optimum manipulative variable from the state variable, reference, and disturbance, including their differential and integral values. In the case of the air conditioner, the

0272-1732/93/1000-0037$03.00 © 1993 IEEE

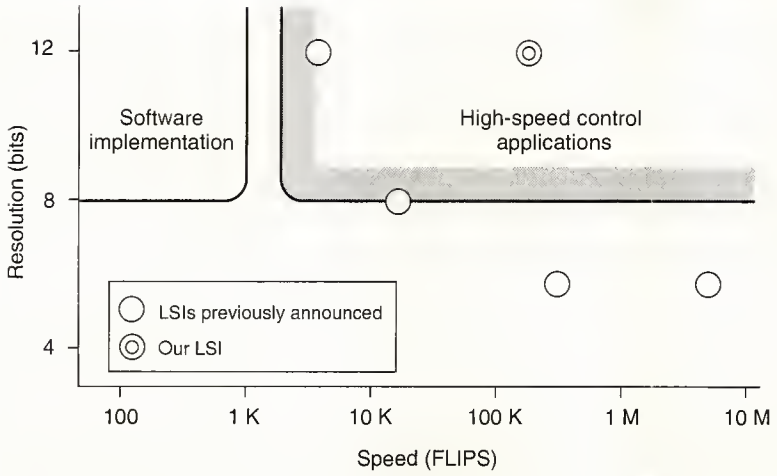


Figure 1. Comparing fuzzy inference performance levels.

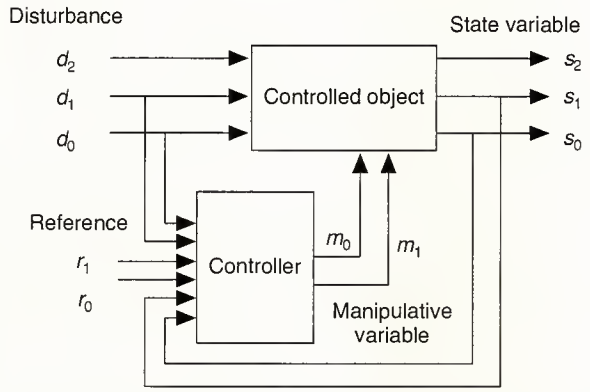


Figure 2. Fuzzy inference control system.

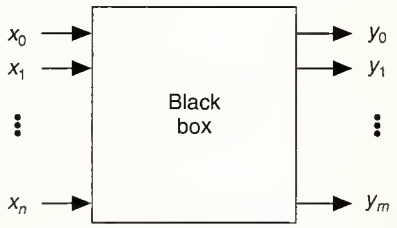


Figure 3. Controller as black box.

manipulative variable would correspond to the motor power.

The key issue for the controller is how to determine the manipulative variable. From this point of view, we can look at the controller as a black box, as shown in Figure 3. At the black box, only the relation between input and output matters. In a conventional controller, a linear or nonlinear function most often defines the relation. Therefore, we calculate output analytically from input according to the function. In a control using fuzzy inference, fuzzy rules define the relation, and output is inferred from input according to the fuzzy inference mechanism.

Fuzzy inference rules. As an example, we can express rules as follows when the number of inputs is n and number of outputs is m:

If $x_0=A_0$ and $x_1=A_1$ and, ..., and $x_n=A_n$, then $y_0=P_0$, $y_1=P_1$, ..., $y_m=P_m$.

If $x_0=B_0$ and $x_1=B_1$ and, ..., and $x_n=B_n$, then $y_0=Q_0$, $y_1=Q_1$, ..., $y_m=Q_m$.

Here $x_0, \ldots, x_n$ are inputs and $y_0, \ldots, y_m$ are outputs. $A_0, \ldots, A_n$, $B_0, \ldots, B_n$, $P_0, \ldots, P_m$, and $Q_0, \ldots, Q_m$ are fuzzy sets. The part of the rule ahead of "then" is called the "antecedent," and the part after it is called the "consequent." We usually define the fuzzy set as a function called the membership function; see Figure 4. In the figure, the horizontal axis represents the input or output. The vertical axis represents the grade, which ranges from 0 to 1 when normalized.

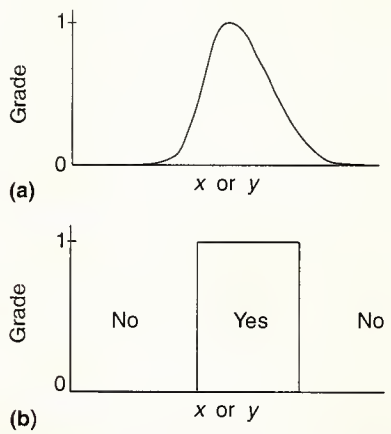


Figure 4. Membership function: fuzzy set (a) and crisp set (b).

Fuzzy theory

Fuzzy theory itself is not "fuzzy" at all, but rather a rigorous mathematical theory. The term *fuzzy theory* is a generic one that encompasses the theories of fuzzy set, fuzzy logic, and fuzzy measure. Fuzzy logic and fuzzy measure theories were derived from fuzzy set theory as established by Lotfi A. Zadeh in 1965.[1]

Fuzzy set theory. In classical set theory, an element is either a member of a set or not (crisp set). In fuzzy set theory, elements can be partially a member of a set (fuzzy set).

Let's take temperature as a universal set. In this case, each temperature—20°C or 27°C—is an element. Figure A shows an example of fuzzy and crisp sets for hot temperatures and temperatures higher than 25°C. In both cases, each element has a value called membership or grade that indicates to what degree the element belongs to the set. While the crisp set takes only two values—complete truth (grade = 1, if normalized) and complete falseness (grade = 0)—the fuzzy set takes continuous values from 0 to 1.

Fuzzy set theory is a generalization of classical set theory because it allows both complete truth and falseness. Therefore, any conclusion derived from classical set theory is also valid with fuzzy set theory.

Fuzzy control. A control that uses fuzzy inference as a means to calculate outputs from inputs is called fuzzy control. Fuzzy inference is based on fuzzy theory, which E.H. Mamdani first applied to control fields in 1974.[2] Control is currently a most active and fruitful field for the application of fuzzy theory, because fuzzy theory is a technically viable and cost-effective discipline. Among the many applications of fuzzy control that have been reported, those for cement kiln control,[3] water quality control,[4] and the automatic train operation system of the Sendai subway[5] are especially well known.

For a fuller historical perspective of fuzzy control, consult Lee[6,7] and Brubaker.[8,9]

1. L.A. Zadeh, "Fuzzy Sets," *Information Control*, Vol. 8, 1965, pp. 338-353.
2. E. H. Mamdani and S. Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," *Int'l J. Man-Machine Studies*, Vol. 7, 1975, pp. 1-13.
3. L.P. Holmblad and J.J. Ostergaard, "Control of a Cement Kiln by Fuzzy Logic," in *Fuzzy Information and Decision Processes*, M.M. Gupta and E. Sanchez, eds., North-Holland, Amsterdam, 1982, pp. 389-399.
4. O. Yagishita, O. Itoh, and M. Sugeno, "Application of Fuzzy Reasoning to Water Purification Process," in *Industrial Applications of Fuzzy Control*, M. Sugeno, ed., North-Holland, Amsterdam, 1985, pp. 19-40.
5. S. Yasunobu and S. Miyamoto, "Automatic Train Operation by Predictive Fuzzy Control," in *Industrial Applications of Fuzzy Control*, M. Sugeno, ed., North-Holland, Amsterdam, 1985, pp. 1-18.
6. C.C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller—Part I," *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 20, No. 2, 1990, pp. 404-418.
7. C.C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller—Part II," *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 20, No. 2, 1990, pp. 419-435.
8. D. Brubaker, "Fuzzy-Logic Basics: Intuitive Rules Replace Complex Math," *EDN*, June 18, 1992, pp. 111-116.
9. D. Brubaker, "Fuzzy-Logic System Solves Control Problem," *EDN*, June 18, 1992, pp. 121-127.

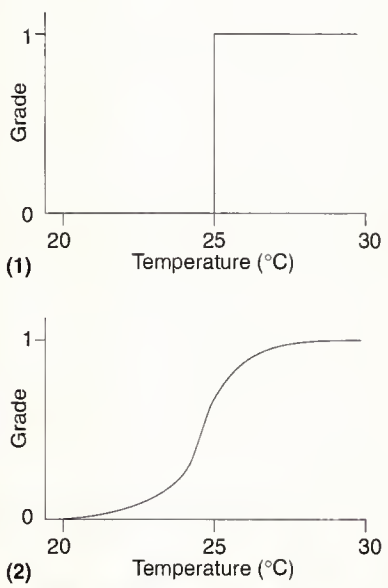


Figure A. Crisp set (1) and fuzzy set (2).

Investigators have proposed several methods for performing fuzzy inference. We will describe the min-max-centroid method, the most popular one. For simplicity's sake, let's consider the case with two inputs, two rules, and one output. The rules are

rule 0: If $x_0 = A_0$ and $x_1 = A_1$, then $y = P$
rule 1: If $x_0 = B_0$ and $x_1 = B_1$, then $y = Q$

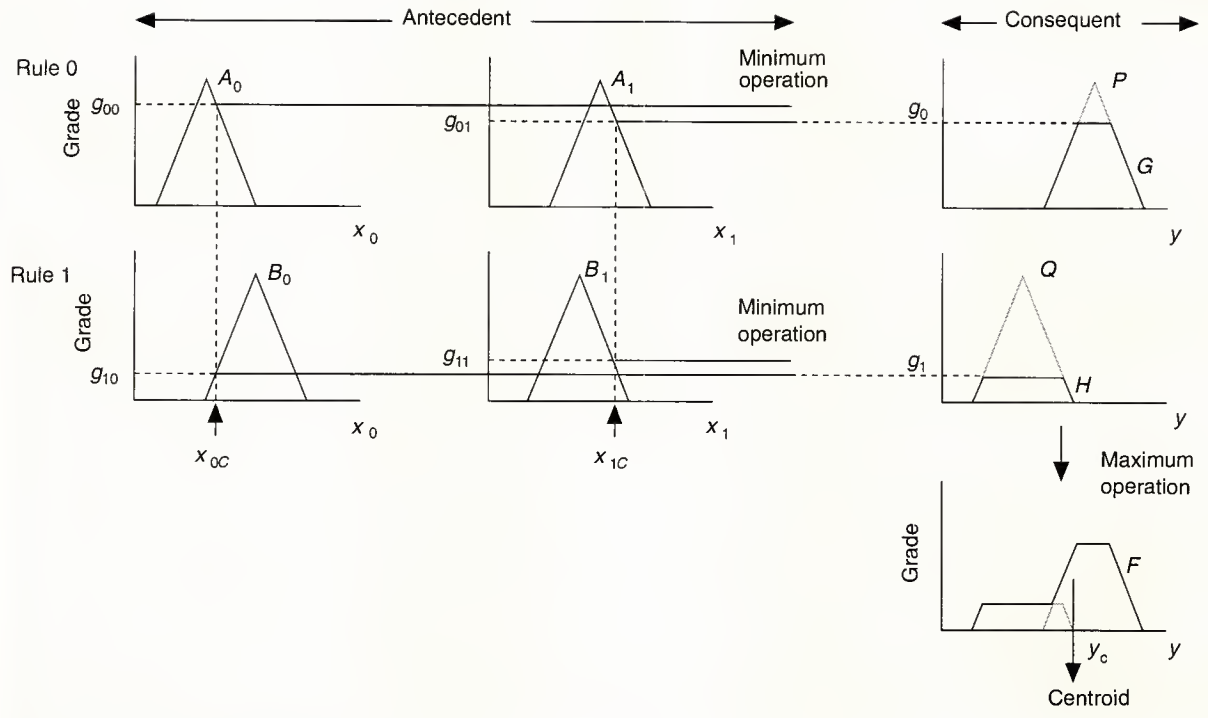


Figure 5. Fuzzy inference schematic.

Figure 5 shows the schematic of the fuzzy inference that results from these rules. As shown, six functions represent membership functions: A_0, A_1, P, B_0, B_1, and Q. Also, x_{0C} and x_{1C} represent the values of input x_0 and x_1, respectively, from which the inference is performed. y_c represents the inferred value as output y.

As the first step of the inference, we obtain the value g_{00}, which represents the grade of the equation $x_0 = A_0$, by calculating the value of the function A_0 when $x_0 = x_{0C}$. We obtain the value g_{01}, representing the grade of the equation $x_1 = A_1$, in the same way.

For the semantic *and* operation, we perform the minimum operation between both values by selecting the lesser. As a result, we obtain g_0, the grade of rule 0 and also g_1, the grade of rule 1. These comprise the antecedent processing.

Consequent processing involves obtaining the function G by lopping off the part greater than value g_0 from function P. We obtain function H in the same way. To perform the maximum operation, we select the greater values of both functions at each value of y. From this operation, we obtain function F, or the resultant membership function.

Defuzzification, the final processing of the inference, entails calculating the centroid, or center of gravity, of the function F. The result of the defuzzification, y_c, becomes the result of the inference as the value of output y.

The fuzzy concept. Linguistic expressions such as cool, warm, or hot are fuzzy concepts. The fuzzy set is the mathematical expression of the fuzzy concept. The grade indicates to what degree the set includes input or output values. In conventional theory, the degree takes only two values—yes or no. This is the crisp set. In fuzzy theory the set has a continuous value. Figure 4 illustrates this distinction.

However, this recognition is rather complex. If we are considering only fuzzy inference, it may be easier to view the rule as follows:

> If x_0 matches A_0 and x_1 matches A_1 and, ..., x_n matches A_n, then output Q as y.

In this case, A_0, ..., A_n specifies the input range. The grade obtained by substituting current values x_0, ..., x_n for A_1, ..., A_n indicates the degree of matching. The antecedent specifies the region of the input n-dimensional space composed of x_0, ..., x_n. The grade of the rule obtained by the *and* operation indicates the degree that current x_0, ..., x_n values match the region. The Q specifies the value to be output when x_0, ..., x_n values match the region. This specifies only a scalar value, though we define it as a fuzzy set. We can regard the calculation of the inference as a way to get a weighted average of values designated in the consequent of each rule, using the grades in the

antecedent of each rule as weighting values.

In any rule, the antecedent specifies a region of the input space, while the consequent specifies the output value when current inputs match the region. The rule set then is a lookup table; fuzzy inference is an interpolation of the table. The main difference from conventional lookup tables is that fuzzy logic defines the input region not crisply but fuzzily and uses a special mechanism called fuzzy inference for the interpolation of the table.

The calculation of a centroid is complex. To simplify the process, we sometimes use singleton functions. With these, consequent membership function does not designate a range but a value. Using these functions, the result of the inference is not so different from cases that use an ordinary function, unless redundancy of the rules exists.

The min-max-centroid method can eliminate the redundancy of the rule set. Consider a rule set having two rules that have the same or nearly the same meaning. Both rules have the same or close consequent membership functions. By calculating the simple weighted average, we duplicate the weighting operation for each rule. Using the min-max-centroid method eliminates this duplication, because the maximum operation selects only one value. At this point, the singleton function is at a disadvantage because it can eliminate redundancy only when consequent membership functions are exactly the same. In the same way, the minimum operation at the antecedent can eliminate the redundancy between the equations (*and* term).

Rule 0 : if $x_0=A_0$ and $x_1=A_1$ and $x_2=A_2$ then $y=P_0$
Rule 1 : if $x_0=B_0$ $x_2=B_2$ then $y=P_0$
Rule 2 : if $x_1=C_1$ and $x_2=C_2$ then $y=P_1$
Rule 3 : if $x_0=D_0$ and $x_1=D_1$ then $y=P_2$
Rule 4 : if $x_0=E_0$ and $x_2=E_2$ then $y=P_2$
Rule 5 : if $x_0=F_0$ and $x_1=F_1$ and $x_2=F_2$ then $y=P_2$
Rule 6 : if $x_1=G_1$ and $x_2=G_2$ then $y=P_3$
Rule 7 : if $x_0=H_0$ then $y=P_3$

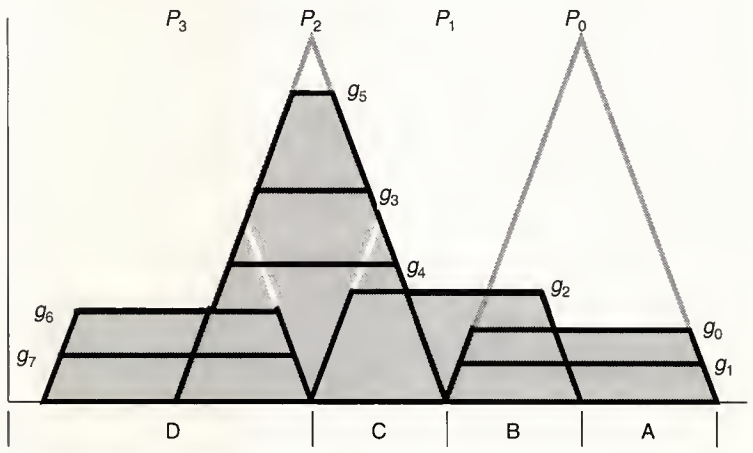


Figure 6. Sample rule set and resultant membership function.

Execution of inference

Next, we want to further investigate the fuzzy inference by exploring antecedent and consequent processing, membership function calculation, and inference reliability.

Antecedent processing. Figure 6 shows an example of a rule set and its resultant membership function. The rule set is composed of rules 0 to 7. The grades of these rules have an assumed value of g_0 to g_7. As shown, only g_0, g_2, g_5, and g_6 affect the inference. These values are the largest grades among the rules that have the same consequent membership function. This is a characteristic of the min-max-centroid method.

We can calculate grades g_0, g_2, g_5, and g_6 as shown in Figure 7 (next page). Here min( ) and max( ) represent minimum and maximum operation, while min register and max register represent the accumulator into which the result of the minimum operation and maximum operation are stored.

Consequent processing. Assume that the resultant membership function $f(y)$ is defined by the discrete values of $f(0), f(1), f(2), \ldots, f(n-1), f(n)$. In this case, the centroid of $f(y)$ is defined with the quotient of the division P by Q, where P and Q are defined as follows:

$$P = f(0) * 0 + f(1) * 1 + \ldots + f(n-1) * (n-1) + f(n) * n$$
$$Q = f(0) + f(1) + f(2) + \ldots + f(n-1) + f(n)$$

Here, we can express P as follows:

$$\begin{aligned} P = & \{f(n)\} \\ & + \{f(n) + f(n-1)\} \\ & + \ldots \\ & + \{f(n) + f(n-1) + \ldots + f(2)\} \\ & + \{f(n) + f(n-1) + \ldots + f(2) + f(1)\} \end{aligned}$$

We can define P_j and Q_j as follows:

$$\begin{aligned} P_j = & \{f(n)\} \\ & + \{f(n) + f(n-1)\} \\ & + \ldots \end{aligned}$$

```
      (initialization)
 0    0 -> max reg;  255 -> min reg
      (rule 0)
 1    min(A0(x0), min reg) -> min reg
 2    min(A1(x1), min reg) -> min reg
 3    min(A2(x2), min reg) -> min reg
 4    max(min(A2(x2), min reg), max reg) -> max reg
 5    255 -> min reg
      (rule 1)
 6    min(B0(x0), min reg) -> min reg
 7    min(B2(x2), min reg) -> min reg
 8    max(min(B2(x2), min reg), max reg) -> max reg
 9    255 -> min reg;  0  -> max reg
      (rule 2)
10    min(C1(x1), min reg) -> min reg
11    min(C2(x2), min reg) -> min reg
12    max(min(C2(x2), min reg), max reg) -> max reg
13    255 -> min reg;  0  -> max reg
                 .
                 .
                 .
      (rule 6)
27    min(G1(x1), min reg) -> min reg
28    min(G2(x2), min reg) -> min reg
29    max(min(G2(x2), min reg), max reg) -> max reg
30    255 -> min reg
      (rule 7)
31    min(H0(x0), min reg) -> max reg
32    max(min(H0(x0), min reg), max reg) -> max reg
33    255 -> min reg; 0 -> max reg
```

Figure 7. Grade calculation schematic

$$+ \{f(n) + f(n-1) + \ldots + f(j+1)\}$$
$$+ \{f(n) + f(n-1) + \ldots + f(j+1) + f(j)\}$$
$$Q_j = f(n) + f(n-1) + f(n-2) + \ldots + f(j)$$

We can then express P as follows:

$$P = Q_n + Q_{n-1} + Q_{n-2} + \ldots + Q_2 + Q_1$$

Since Q_j is the temporary result in calculating Q, the following steps give us P and Q:

- Set $j = n$, $Q_j = 0$, and $P_j = 0$.
- Add Q_j to P_j, and $f(j)$ to Q_j.
- Subtract 1 from j.
- Repeat steps 2 and 3 until j becomes negative.
- Q is obtained as Q_j and P is obtained as P_j.

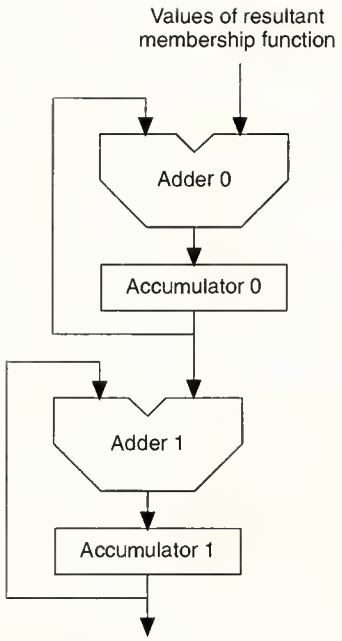


Figure 8. Two-stage adder/accumulator.

A two-stage addition/accumulation can easily perform these steps either through software or by using the hardware shown in Figure 8.

In the example shown in Figure 6, we can obtain P and Q by using the algorithm composed of the following nine processes. This is the algorithm we employed in designing our fuzzy inference processor. It lets us pipeline both the antecedent processing (rule calculation and minimum/maximum operation) and the consequent processing (accumulation).

1. Arrange the rules in the descendent order of the values of their consequent membership functions. (In Figure 6, rules 0 to 7 are already arranged according to the larger functions P_0 to P_3).
2. Calculate grade g_0 and g_1 and perform the maximum operation between these values.
3. Perform a two-stage accumulation in area A of the function $f_0(y)$ expressed as follows:

$$f_0(y) = \min\{P_0(y), g_0\}.$$

4. Calculate grade g_2.
5. Perform a two-stage accumulation in area B of the function $f_1(y)$ expressed as follows:

$$f_1(y) = \max[\min\{P_0(y), g_0\}, \min\{P_1(y), g_2\}].$$

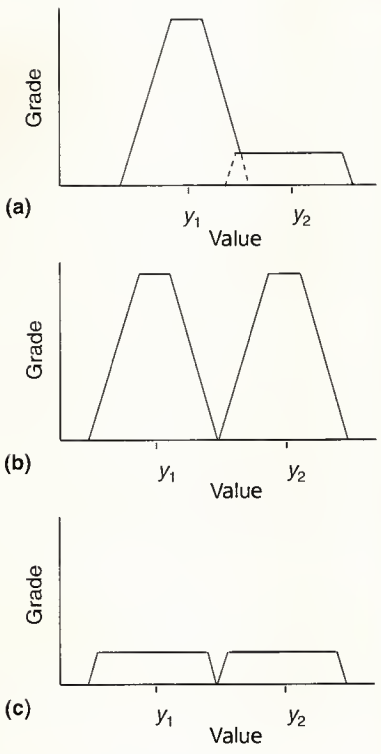


Figure 9. Resultant membership function: no conflict (a); conflict, with large variance (b); conflict, no rule has large grade (c).

6. Calculate grades g_3, g_4, and g_5 and perform the maximum operation among these values.
7. Perform a two-stage accumulation in the area C of the function $f_2(y)$ expressed as follows:

$$f_2(y) = \max[\min\{P_1(y), g_2\}, \min\{P_2(y), g_5\}].$$

8. Calculate grades g_6 and g_7 and perform the maximum operation between these values.
9. Perform a two-stage accumulation in the area D of the function $f_3(y)$ expressed as follows:

$$f_3(y) = \max[\min\{P_2(y), g_5\}, \min\{P_3(y), g_6\}].$$

Membership function calculation. For a fast inference it is also important to calculate the value of membership function. Using a lookup table may be best when software performs the inference, even though it requires a large memory. The literature cites an example of the hardware implementation.[4]

Inference reliability. Though the conventional fuzzy inference mechanism has no facility to evaluate the reliability of the inference, practical applications need such a reliability evaluation. The following two mechanisms serve this purpose.

Calculation of variance. The resultant membership functions shown in Figures 9a and 9b have the same centroid value. In both functions, the rule causing the left trapezoid asserts to output the value y_1, while the rule causing the right trapezoid asserts to output the value y_2. At the function in Figure 9a, no conflict exists because the grade or "closeness to truth" of the former rule is large but the grade of the latter is small. On the other hand, at the functions in Figure 9b, both grades are great, so both rules conflict with each other. Checking the variance can distinguish these cases. A three-stage adder/accumulator unit can establish the variance.[3]

Calculation of maximum grade. Comparing the two resultant membership functions shown in Figure 9a and 9c, we see that the maximum value of the former is relatively great, but the value of the latter is very small. The result from the former thus has considerable "closeness to truth," but the result of the latter has little. We can easily check this by estimating the maximum grade of the rule.

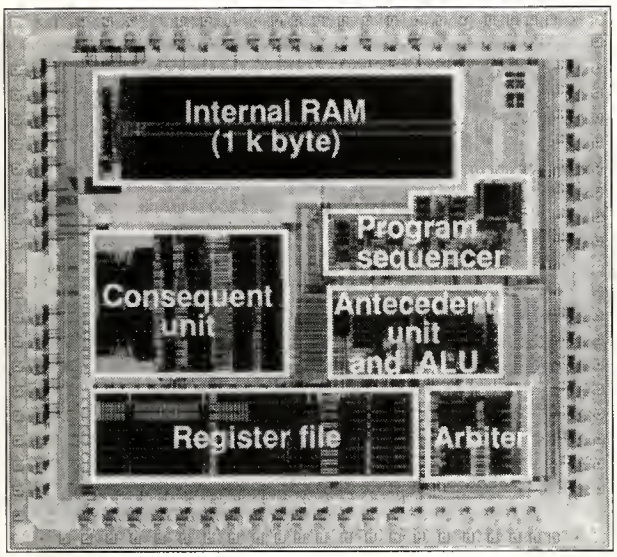


Figure 10. Photomicrograph of fuzzy inference processor.

Fuzzy inference processor

Now that we understand some of the concepts underlying it, we can move on to describe the actual fuzzy inference processor we have created.

Performance. Figure 10 shows a photograph of the fuzzy inference processor and Table 1 (next page) gives its specifications.[5] The processor can perform an inference at 200 KFLIPS with 12-bit input resolution and 16-bit output resolution. For these calculations, we use a typical inference having about

Table 1. Performance of fuzzy inference processor.

Parameter	Specification
Inference speed	200 KFLIPS (20 rules with 2 inputs/1output)
Input resolution	12 bits
Output resolution	16 bits
Membership function shape variations	About 30,000
Number of rules	More than 15,000 (2inputs/ 1 output)
Clock rate	20 MHz
Power supply	+5V
Package	80-pin package

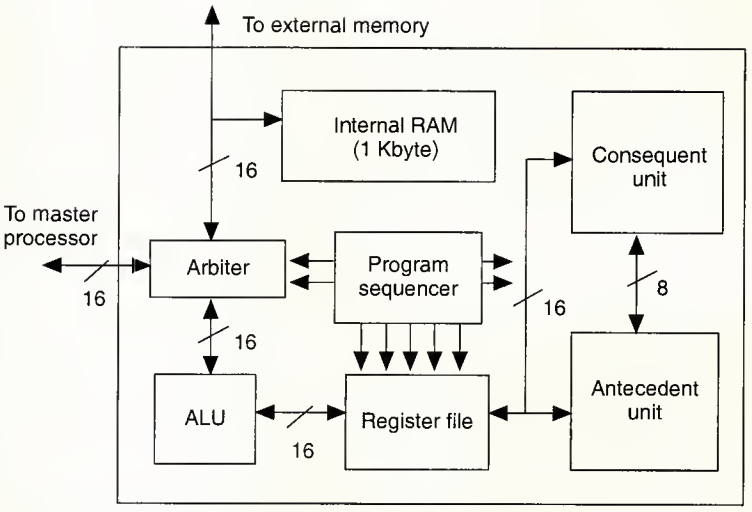


Figure 11. Fuzzy inference processor block diagram.

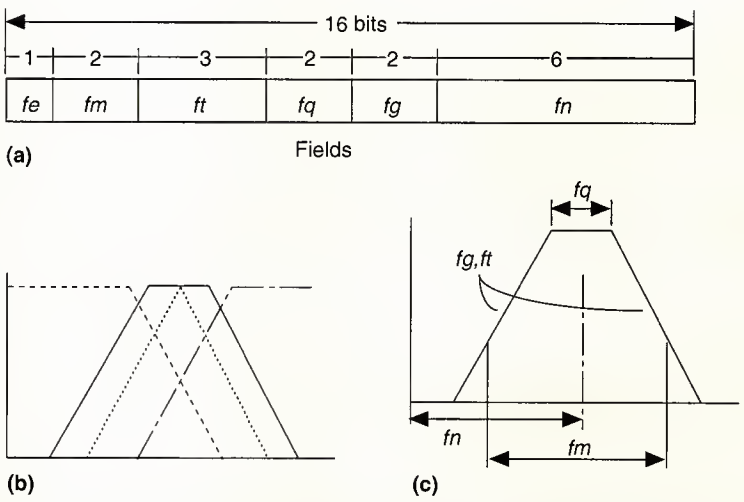


Figure 12. Membership function and function parameter: 16-bit function parameter (a), function shapes (b), and fields (c).

20 rules with two to four inputs and a single output. We determined these performance levels by researching the engine control as follows.

For automobile engine control, inhaled air mass is the most significant input. Currently, designers sample this input at 8 bits; they will sample it at 10 to 12 bits in the near future. An eight-cylinder engine running at 10,000 rpm is ignited 1,333 times per second. In determining ignition delay timing, we find that performing several inferences in the interval between ignitions will require an inference speed of greater than 10 KFLIPS. That makes 100 KFLIPS a reasonable target for the inference in light of other outputs such as fuel mass.

Inference processor hardware. See Figure 11 for a block diagram of this processor. The antecedent and consequent units are the special hardware added for the fuzzy inference. The arithmetic logic unit, register file, internal RAM, and program sequencer constitute a conventional microprocessor. The processor has two processor modes: master (for stand-alone use) and slave (for coprocessor use). Using the arbiter, the master processor can access the register files, internal RAM, and external memory without resorting to off-chip hardware.

The data width for processing in the ALU, internal RAM, and register file is 16 bits, a rate that was determined to preserve conformity with the master processor and efficiency of the instruction code. Antecedent processing, performed in the antecedent unit, varies depending on the number of rules and inputs. Accumulation for the centroid calculation, performed in the consequent unit, is usually realized by a fixed sequence. According to our design, the program composed of the rule instructions controls the antecedent unit, while the hardwired logic controls the consequent unit.

Membership function generator. The membership function generator calculates the value of the membership function. As shown in Figure 12b, this dedicated hardware generates four kinds of function shapes based on a trapezoid. The 16-bit data called the function parameter specifies the function (see

Figure 12a). The parameter consists of six fields: *fn* for specifying the center position of the function, *fq* for the length of the top side if the function is trapezoid, *fm* for the scale factor of the horizontal direction, *fg* for specifying the inclination, and *ft* for both specifying the function-shape and modifying the inclination of the oblique sides if the function shape is trapezoidal or triangular. Using the function parameter, more than 30,000 variations of membership function can be generated. The optional *fe* field allows for changing the shape of the oblique sides (for example, from straight line to S-shaped) by referring to a lookup table. A microprogram included in the program sequencer controls this change.

Antecedent unit and rule instruction. The antecedent unit performs the grade calculation of the rule when the program designates a rule instruction. Included in the register file are 16 data registers that hold the input values of the inference which were stored there in advance. These instructions fall into two categories: crule (pronounced see-rule), which triggers the accumulation for the centroid calculation, and rule, which does not. Figure 13 compares the formats of these instructions.

Each instruction consists of an opcode and a register list followed by the membership function parameters. The register list represents the input used for the inference. Paired with each data register is each corresponding bit in the register list; setting the corresponding bit to 1 designates an arbitrary data register. The membership function parameters individually specify the shape of the membership functions. The number of the parameters therefore corresponds to the number of bits that are each set to 1 in the register list of the rule instructions.

Crule instructions have one additional function parameter: The last parameter designates the consequent membership function. In the crule instruction, the opcode includes the accumulation constant—represented as cc in Figure 13—that specifies the repeat count of accumulation. The centroid calculations use these parameters and constant values.[4]

Figure 14 shows the block diagram of the antecedent unit. The membership function generator generates the values of the membership function designated by the input value and function parameter. The min operator performs the minimum operation between the function value and the value in the min register, which is initially set to the maximum value of the grade at the end of the previous rule or crule instruction execution. The min register stores the result.

The max operator performs the maximum operation between the result of the minimum operation and the value in the max register, which is cleared to 0 at the end of the previous crule instruction execution. The max register stores the result. After performing the maximum operation, the antecedent unit sets the min register to the maximum value of the grade. When executing a rule instruction, the unit performs no additional operations, and the result of the execution remains in the max register.

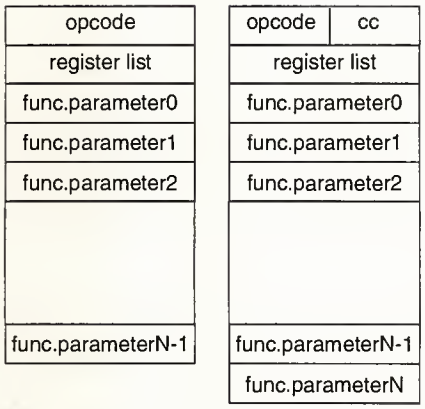


Figure 13. Rule instruction: rule (a) and crule (b).

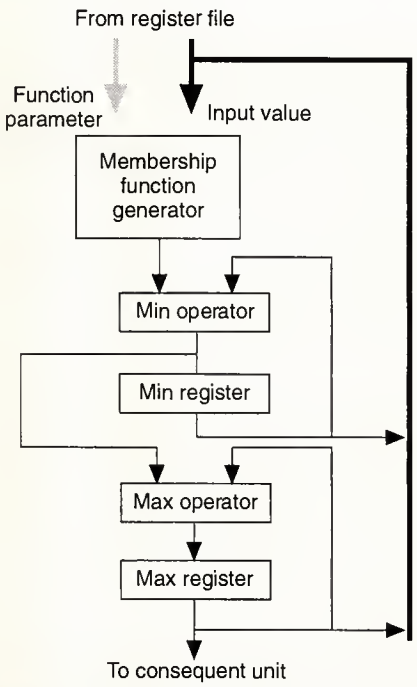


Figure 14. Antecedent unit block diagram.

When executing a crule instruction, the antecedent unit performs an additional operation. It transfers the last function parameter to the consequent unit along with the value in the max register and the accumulation constant in the instruction register. After the transfer, the unit clears the max register to 0.

Consequent unit. This unit is divided into the min/max

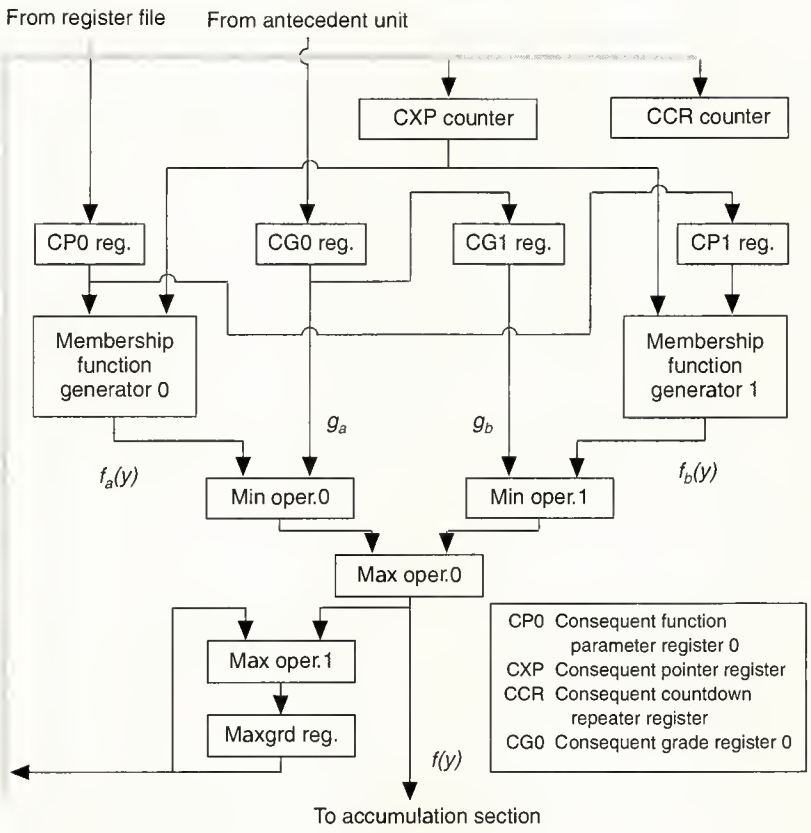


Figure 15. Min-max section block diagram.

```
1   inicp    MaxY
2   rule     [r0,r1,r2],A0,A1,A2
3   crule    [r0,r2],B0,B2,P0,CnstA
4   crule    [r1,r2],C1,C2,P1,CnstB
5   rule     [r0,r1],D0,D1
6   rule     [r0,r2],E0,E2
7   crule    [r0,r1,r2],F0,F1,F2,P2,CnstC
8   rule     [r1,r2],G1,G2
9   crule    [r0],H0,P3,CnstD
10  cntrd    OfstY
11  vrns     ThvV
12  cmpgrd   ThvG
```

Figure 16. Sample inference program.

section and the accumulation section. Figure 15 (next page) shows the min/max section. The following equation expresses $f(y)$, which is the output of the max operator 0:

$$f(y) = \max\{\min\{f_a(y), g_a\}, \min\{f_b(y), g_b\}\}$$

where $f_a(y)$ and $f_b(y)$ are the membership functions designated by the parameters in the CP0 and CP1 registers, g_a and g_b are the grades stored in the CG0 and CG1 registers, and y represents the value of the CXP counter.

The max operator 1 performs the maximum operation between the value in the maxgrd register and the value of $f(y)$. The consequent unit clears the maxgrd register to 0 in advance of the inference, and stores it with the value selected by the max operator 1, during the calculation of $f(y)$.

The accumulation section contains the three-stage adder/accumulator unit. It accumulates the value of the resultant membership function calculated in the min/max section. See Nakamura et al.[4] for details.

Inference program. Providing the rule instruction makes this program quite simple. Figure 16 shows an inference of the rule set given in Figure 6. This example assumes that inputs of $x_0, \ldots, x_2$ are already stored into data registers of $r0, \ldots, r2$.

In the figure, the INICP instruction initializes the registers. The max register, CG0 register, and three accumulators included in the three-stage adder/accumulator unit are initialized to 0. MaxY, which is stored into the CXP counter, represent the maximum value of y. Rule instructions and crule instructions calculate the grade of the rules and the resultant membership function according to the algorithm described earlier. In these instructions, registers in brackets represent the register list. $A_0, \ldots, H_0$ represent the function parameters of antecedent membership functions. $P_0, \ldots, P_3$ represent the function parameters of consequent membership functions. CnstA, ..., CnstD represent the accumulation constants. These values are the widths of the area $A, \ldots,$ area D in Figure 6.

Each rule/crule instruction calculates a grade of the rule and performs the maximum operation between the grade and max register. Only crule instructions trigger accumulation, which they do by transferring the accumulation constant, the value of the max register, and the consequent membership function to the consequent unit. After the transfer, the processor clears the max register to 0.

As shown in Figure 17, the accumulation is pipelined with the execution of the rule instructions.[4] The cntrd instruction calculates the centroid, then adds the value of OfstY to it. OfstY is the offset value that is added to the centroid. The

vrns instruction calculates and checks the variance. ThvV represents the threshold value of the variance. The cmpgrd instruction checks the maximum grade by comparing the maxgrd register with the value of ThvG.

Execution time. The execution time of the rule instruction specifying N data registers is $N+1$ clock cycles. For the crule instruction, it is $N+2$ clock cycles. The total execution time of inicp, cntrd, vrns, and cmpgrd is 44 clock cycles. The accumulation time taken by the three-stage adder/accumulator unit is 64 clock cycles when the maximum value of y is 64. The inference time shown in Figure 16 becomes 73 clock cycles if accumulation is performed in the execution time of the rule instructions. However, in this case, accumulation time may be greater than the execution time of the rule instructions. So, inference time becomes about 100 clock cycles when the maximum value of y is 64.

In the same way, with up to about 20 rules having two to four inputs, the inference time depends on the accumulation time and becomes about 100 clock cycles. This corresponds to 200 KFLIPS performance with a clock of 20 MHz. When the processor performs inferences with more complex rule sets, the inference time depends on the execution time of the rule instruction.

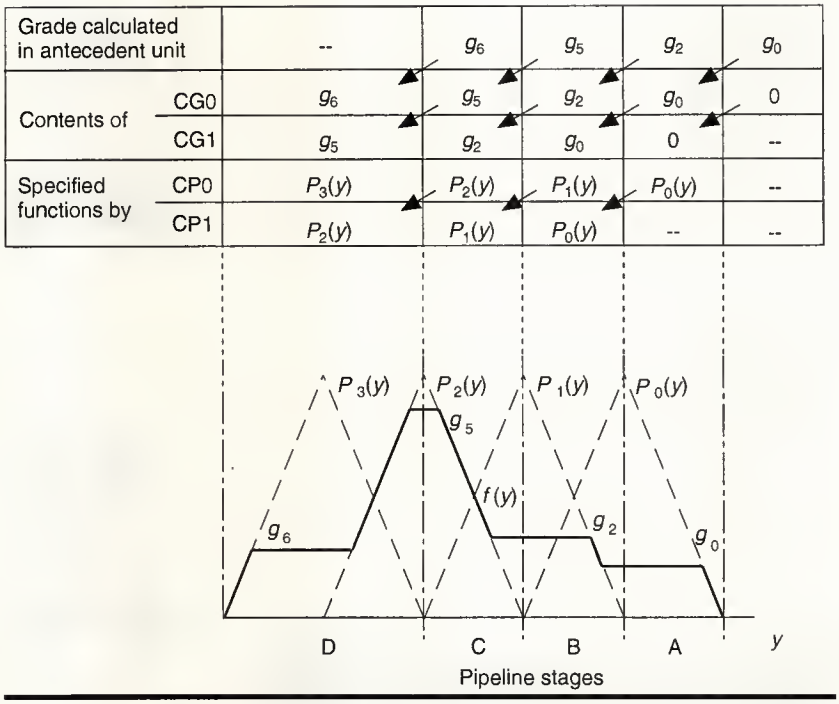


Figure 17. Consequent membership functions and pipelined operation.

Acknowledgments

We gratefully acknowledge Hiroyoshi Komiya, Akira Iwase, and Osamu Tomisawa for their encouragement and advice.

WE CAN LOOK UPON THE FUZZY inference data processing method as an interpolation of a lookup table. It is also a method to define and calculate a nonlinear function. Most significantly, its data processing mechanism can be described by the linguistic expression used in normal speech. The support tools that help to define rule sets or membership functions are important for taking advantage of fuzzy inference. As dedicated LSIs and support tools become available, we believe fuzzy inference will find wider application as it substitutes for conventional data processing.

References

1. L.A. Zadeh, "Fuzzy Set," *Information and Control*, Vol. 8, 1965, pp. 338-353.
2. E. H. Mamdani and S. Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," *Int'l J. Man-Machine Studies*, Vol. 7, 1975, pp. 1-13.
3. H. Watanabe, W. D. Dettloff, and K. E. Yount, "A VLSI Fuzzy Logic Controller with Reconfigurable, Cascadable Architecture," *IEEE J. Solid-State Circuits*, Vol. 25, No. 2, 1990, pp. 376-382.
4. K. Nakamura et al., "A 12-Bit Resolution 200 KFLIPS Fuzzy Inference Processor," *IEICE Trans. Electronics*, Vol. E76-C, No. 5, July 1993, pp. 1102-1111.
5. K. Nakamura et al., "A 12-Bit Resolution 200 KFLIPS Fuzzy Inference Processor," *ISSCC Digest of Technical Papers*, Feb. 1993, pp. 182-183

Kazuo Nakamura is a senior engineer at Mitsubishi Electric's Kita-Itami works. During this project, he worked at its LSI laboratory, where he was engaged in application engineering of LSI and design engineering of general-purpose processors, fuzzy inference processors, and their support software. Nakamura received BS and MS degrees in electrical engineering from the Tokyo Institute of Technology.

Narumi Sakashita is a research engineer at Mitsubishi's Semiconductor Research Laboratory. His interests include design engineering of digital signal processors and fuzzy inference processors.

Sakashita received a BS in electrical engineering from Kansai University, Osaka, Japan. He is a member of the Institute of Electronics, Information, and Communication Engineers.

Yasuhiko Nitta is a research engineer at Mitsubishi's Semiconductor Research Laboratory where he has been engaged in the design engineering of general-purpose processors and fuzzy inference processors.

Nitta received a BS in electrical engineering from Waseda University, Tokyo, Japan.

Ken'ichi Shimomura is a research engineer at Mitsubishi's Semiconductor Research Laboratory where he has been engaged in the design engineering of general-purpose processors and fuzzy inference processors.

Shimomura received BS and MS degrees in physics from Osaka University.

Takeshi Tokuda is manager of the R&D planning section of Mitsubishi's System LSI Laboratory. His interests include development of CMOS logic LSI and digital signal processor.

Tokuda received BS and MS degrees in communications engineering from Osaka University. He was also a visiting scholar at Stanford University, where he worked on VLSI processor design. He is a member of the Institute of Electronics, Information, and Communication Engineers.

Direct questions concerning this article to Ken'ichi Shimomura, Mitsubishi Electric Corporation, Semiconductor Research Laboratory, 4-1 Mizuhara, Itami, Hyogo 664, Japan; or at simomura@micro.lsi.melco.co.jp.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate numbe on the Reader Service Card.

Low 156 Medium 157 High 158

Special Report:
HDTV Research in Japan

High-definition television has generated a great deal of interest in a variety of fields in recent years. While research in the United States and Europe has focused on a digital approach, Japan has already developed an analog system. It is now applying its knowledge toward development of digital HDTV.

David K. Kahaner

US Office of Naval Research, Far East

kahaner@cs.titech.ac.jp

[David Kahaner is on assignment with the US Office of Naval Research. He generally comments on activities in East Asia for inclusion in the Software Report column. Since we felt readers would be interested in a detailed description of Japan's research on high definition television, we also offer this special report. His comments are his own; they do not express any official policy.–Ed.]

Most research on high-definition television in the West has focused on digital, rather than analog, technology. As the digital method is more flexible, many in the field believe that these efforts will ultimately succeed in making digital HDTV products more successful and profitable than those using analog technology. While digital HDTV is not currently available, Japan has developed an HDTV system based on analog technology (known as Hivision) that has been in use since 1991. As there are many technical problems shared by both types of systems, research in digital HDTV has also been active in Japan. The Japanese have estimated that digital receivers will not become fully integrated until the next century. Until then, Japanese manufacturer's have their potentially large market to themselves.

I recently participated in NHK's annual open house. Thousands of visitors filed through displays highlighting interesting new research and prototype activities, many of which centered on technology aimed at HDTV. Although the crowds made serious discussion about any of the projects difficult, abstracts for each display were available. A look at some of the more interesting displays provides an overview of Japanese research in this area.

Broadcast, transmission, and reception

Research to develop high-quality systems for broadcast, transmission, and reception forms the core of NHK's work on Hivision. I have chosen several projects in each of these areas to highlight.

Attaining high picture quality through MUSE. MUSE is a transmission system that enables broadcasting on a single channel, the same as with current satellite broadcasts, using bandwidth compression technology. Experimental Hivision broadcasts using the MUSE method now take place eight hours a day. Hivision video signals equivalent to five times the current TV signals provide Hivision video to viewers throughout the country.

Integrated service digital broadcasting. Analog waves currently carry TV and audio signals, with digital waves likely in the future. Compression technology makes multichannel TV with a single broadcasting wave possible. ISDB is a broadcast system capable of transmitting various digitized services by integrating multilayers on a single radio wave.

0272-1732/93/1000-0049$03.00 © 1993 IEEE

Development of a system to allow viewing of 3D images without special glasses is a basic research theme.

Joining Hivision with computers will allow additional combinations of multifunctional media and information services.

Digital broadcasting on land. Weakening of reception signals by multipath phased propagation represents a significant obstacle to receiving land broadcasting waves on mobile bodies, such as automobiles. Conventional methods of transmission make complete duplication of signals by the receivers difficult. Therefore, a new method of transmission based on digital audio broadcasting is under consideration. This approach involves multiplexing multiple stereo programs within a wide band modulated through orthogonal frequency division multiplexing. In addition, this method also allows for effective mobile transmissions.

42-GHz-band Hivision digital FPU. An analog type Hivision repeater transmission unit using a 42-GHz-band radio wave is already in use. However, rainfall greatly weakens the 42-GHz band, limiting transmissions using this method to relatively close ranges. Accordingly, researchers developed the 42-GHz-band Hivision FPU to expand the range of high picture quality through digital transmissions.

Demand-access optical CATV system. In the near future, multichannelization of satellite broadcasts will result in high growth of new broadcast services, led by Hivision. Researchers are investigating fiber-to-the-home broadcasts that would prove suitable for satellite broadcasting.

Development of a high-quality, inexpensive multichannel video distribution service, which makes the FM-FDM type of demand-access optical CATV system possible, is complete.

Stratified digital transmission. In broadcasts using the digital transmission method, reception often degrades suddenly due to obstacles and strong rainfall. When applying digital transmissions to broadcasting, it is preferable for the degradation in reception quality to occur slowly. The stratified transmission method attempts to prevent sudden degradation.

Hivision reception quality measuring devices. Investigators have developed devices that measure high frequency and MUSE signal characteristics to evaluate Hivision reception quality.

Mobile receiving system carried aboard aircraft. Since satellite broadcasting covers wide areas, reception in mobile bodies provides an ideal application. Development of a mobile receiving system for ships, trains, sightseeing buses, and cars is complete. The current project involves development of a system for aircraft to allow maximum use of the wide-area feature of satellite broadcasting.

Portable satellite news-gathering system using flat-surface antennae. Developments in digital technology have led to decreased size for high-performance SNG systems. As a result, researchers expect phenomenal improvements in system operability and further perfection of broadcasting program material. Investigators are proceeding with the development of a portable SNG employing flat-surface antennae and solid-power amplifiers, allowing convenient transportation of the system.

Mobile receivers for FM multiplex broadcasting. Digital signals are multiplexed via pauses in the voice signals of existing stereo broadcasting. Efficient use of frequencies allows assembly of an inexpensive system. Broadcasting via FM multiplex mobile receivers permits travelling vehicles to receive a variety of infomation (such as traffic conditions, news, weather, and stock prices) in real time.

Three-dimensional technology

Television in three dimensions has held the imagination of the entertainment industry for a long time. NHK hopes to couple 3D television with Hivision, and is developing several products with this goal in mind.

3D Hivision without glasses. Development of a system to allow viewing of 3D images without special glasses is a basic research theme in the area of 3D television. In line with this goal, researchers have developed the 70-type liquid crystal projection 3D Hivision display that improves the quality of 3D images as well as a simulator that electrically reproduces multieye 3D images, using a biconvex lens.

Small Hivision camera for 3D photography. A small camera with the properties of easy operation and mobility would allow the use of 3D television in a wide variety of fields (medicine, for example). In addition, a camera with a short distance between the camera lenses (approximately the distance between human eyes) would produce more natural 3D images. Investigators have developed such a Hivision camera and mount.

Audio image distance control for 3D television. The stereophotographic impact of 3D television would increase dramatically if the audio images also jumped out three dimensionally. Mixing the high points of sound pressure provides such a sense of distance of audio images (achieved by localizing the sound points in a single point within a vacuum). The recently developed audio image control system has both audio image dimensionality and continuous position shifting.

HARP technology

NHK hopes that use of HARP technology will lead to increased sensitivity in cameras used to shoot Hivision programs.

Hivision super-HARP handy cameras. Improvement in the characteristics of the HARP camera tube would aid development of a supersensitive camera capable of coping with the diversification in Hivision program production. The supersensitive, high-quality camera with a 1-inch, 55-type super-HARP tube (static accumulation/electrostatic deflection) developed last year has already photographed the Aurora.

Current experiments with the new small, 2/3-inch, 55-type super-HARP tube have led to development of a supersensitive Hivision handy camera.

High-sensitivity imaging through intermittent scanning. The shutter speed of an ordinary TV camera averages 1/60 second. By slowing down shutter speed and scanning beams intermittently over a set time period, even dark objects photograph easily. Pairing this approach with the cascading effect (the essence of the HARP tube) produces extremely high sensitivity. By connecting this intermittent scanning adapter to the HARP camera (525-line type), the possibility of photographing very slow moving objects exists.

Camera technology

While transmission and reception of programs is important to the quality of Hivision, even the highest quality equipment is only as good as the image provided. Superior images require superior cameras. NHK's research on camera technology applies to Hivision as well as more general areas of photography.

High-performance Hivision 4-panel-type, charge-coupled device image experiment. To provide more Hivision programs, industry requires a small camera with excellent picture quality and high mobility. To satisfy this demand, researchers must study high-performance image elements along with imaging methods suitable for high resolution. Previous cameras used the tricolor resolution prism. However, by combining a color resolution prism split in two with four 2/3-inch, 1.3-million-dot CCDs, investigators produced a 4-panel-type (quad-CCD) image-testing system. This approach produced superior resolution, sensitivity, and dynamic range.

Intelligent robot cameras. Producing programs effectively often relies on the use of robot cameras, both for broadcasting programs and in news studios. However, the conventional robot cameras cannot follow the movements of objects being photographed. For example, when an object shifts position and moves out of the picture angle, robot cameras cannot correct the picture angle.

However, the intelligent robot camera automatically corrects the picture angle (camera frame). Using image processing technology, smooth camera following and real-time corrections are possible.

Conventional robot cameras cannot follow the movement of objects. The intelligent robot camera automatically corrects the picture angle.

Automatic animal-tracking camera. When shooting the ecology of wild animals, the presence of a photographer becomes a big obstacle for the animal with a strong sense of caution. Problems in obtaining effective images of the animal's natural behavior result. Automatic photography with unmanned cameras sometimes yields good images. However, wild animals often refuse to cooperate with stationary cameras and move away from the field of vision.

Development of the automatic tracking camera aimed to remedy this difficulty. This device captures the animal as it appears. The camera then tracks it, recording its actions as it moves in a wide area, giving us a more realistic record of its behavior. This new technology provides a new realm in animal photography. Furthermore, it would provide an effective surveillance tool if used in crime prevention systems.

Electronically driven, super-high sensitivity camera elements. Past research on various types of super-high sensitivity camera devices for photography at ultra-low lighting levels concentrated on those joining the image intensifier and the CCD, and those using fiber optics. Waves in the fiber plates and the granular condition of the fluorescent surface degrade in level of resolution levels and picture quality. Current research centers on a new, electronically driven device that does not use fiber plates or fluorescent surfaces.

Image composition and picture quality

Current program composition packages do not always meet the quality standards required by Hivision. As a result, a variety of techniques to boost composition quality are under development.

Hivision domain-extracting system. Production of TV programs and movies depends heavily on image composing technology, particularly domain extraction. Until now the Chromaki method has formed the basis of this technology. However, this method requires a special background, known as blue back, which precludes application to natural images. An extracting technology for natural imaging would prove indispensable in program production.

Researchers previously developed a procedure to cut an

A continuous voice recognition system would automate superimposition of voices enunciated by an indefinite number of speakers in broadcast programs.

object from a randomly moving image, automatically describing the object in a general way. This procedure forms the basis of the new extracting system for use with Hivision. It allows the composite processing of program production, and can also construct an image parts database for time-space editing of images in an image production environment (desktop program production).

Standard animated image for evaluation of Hivision systems. In the overall evaluation of picture quality of Hivision systems and equipment, it is essential to use a variety of picture patterns. Different evaluation objectives require different pictures. Furthermore, uniform evaluation results require use of a common picture. Therefore, the industry requires picture standards.

In response to this need, the Broadcast Technology Association is selecting standard pictures. Once compiled and distributed, these picture sets will become the standard Hivision animated pictures used for system evaluation.

Video computer desktop program production. DTPP will improve programming functions, enlarge video displays, improve program quality, and conserve energy. It aims to simplify the process of producing programs and provides features such as editing, fabricating, and mixing. Simplified access to video materials and information such as camera parameters and lighting conditions, all while remaining at one workstation, would result.

Movement compensating sequential scanning conversion. Current TV interlaced scanning methods transmit every other scanning line. Interlaced scanning compresses the bandwidth of TV signals and is useful in effectively broadcasting radio waves. However, because the thin horizontal lines and oblique lines flicker, they interfere with picture quality.

To correct this interference, Clearvision stores received video pictures in memory, intermixes current images with those preceding them, and then displays the images without gaps between scanning lines (sequential scanning conversion display). This process, however, results in only small image quality improvements. A variation mixes a current image with the image just corrected, resulting in higher quality images with less interference to picture quality.

Voice

Television presents many opportunities for the application of voice recognition. As a result, this topic receives a great deal of attention from numerous research groups, including NHK.

Real-time voice speed converter receiving system. Quality voice broadcasting requires a voice speed converter system that automatically converts fast speech to slower speech. The system divides voice inputs into a soundless sector (a pause), voiceless sectors (consonants), and voice sectors (vowels). It then extracts pitch correctly in voice sectors, interpolates wave forms, and extends the speech speed by maintaining the pitch at a fixed level. By controlling the soundless sections, the system extends intervals.

Superimposition through voice recognition. A continuous voice recognition system would automate superimposition of voices enunciated by an indefinite number of speakers in broadcast programs (news, for example).

Listeners base recognition on "knowledge such as grammar." The system recognizes the vowel portion of voice inputs, compares the consonants of each candidate, and outputs the most appropriate recognition results. In concrete terms, the voice to be recognized becomes a point of position within the vector space expressing the frequency characteristics. Vowel recognition occurs according to the distance from the representative point consonants and their proximity to the statistical model (hidden Markov model). In cases of limited usage, the system recognizes units of sentences (units of commonly used clauses) from continuous voice samples. Researchers are currently experimenting with a small-scale system that uses several parallel microprocessors.

Recording and tape technology

Recording technology represents another important area affecting overall program quality. Both video tape recorders and tapes themselves can determine the overall excellence of Hivision.

Reproduction of recordings on high-density vertical magnetic tapes. Smaller and longer duration digital video tape recorders demand the availability of high-performance tapes. Researchers have developed the high-performance cobalt-chromium-tantalum vertical magnetic tape for this purpose. When compared to the conventional stretched magnetic recording, this tape shows superior performance in reproduction of lengthy shortwave recordings.

Linear scanning VTR. The recent increase in the amount of information required to record images ties directly to the development of higher resolution displays and higher quality processing. The desire for mobility coupled with the increase

fueled an interest in investigating improved recording densities and thinner tapes. Research has resulted in a new VTR head scanning method that makes the mechanical system compact and is suitable for ultra-thin tapes.

Rotating optical recording head. Optical magnetic tapes would permit high-density recording, thereby increasing recording capacity. Research continues on development of an optical magnetic tape for use as an ultra-large capacity optical memory for data preservation and 3D pictures.

Displays

When choosing Hivision over regular television, the consumer will often base the decision on the quality of the image displayed. NHK's work in this area has focussed on developing flat-surface televisions using several display technologies.

Hivision TV with a plasma display. Full enjoyment of Hivision video requires a large screen. The weight and volume constraints point to the so called "wall-type television" as most practical. Investigators have selected the plasma display (discharge plasma), which is relatively simple to enlarge, and are proceeding with development a wall-type television of over 50 inches.

High-definition EL panel display. Electro-luminescent panels possess superior characteristics, such as thinness, light weight, self-luminous, and wide field of view. In addition, these all-solid panels show promise as high-definition, flat-surface displays. Researchers have currently developed and test produced delicately detailed pitch display panels with high-intensity EL layers.

Miscellaneous

Many projects under investigation at NHK do not easily fit under a major category. Although their applications generally deal with television and broadcasting, these projects carry on almost independently.

Kite plane. When gathering data about natural and manmade disasters, aerial photographs provide great impact. A kite plane equipped with superior safety features allows photography under conditions normally unsafe for human pilots and photographers.

Liquid crystal lighting elements. Liquid crystal optical modulating elements possess superior characteristics for high rates of transillumination. These devices do not require deflector plates, have rapid response, and easily cover a wide area. Investigators have developed heat resistant liquid crystal optical modulating elements (with movements possible at temperatures in excess of 160°C).

Radio disturbance reduction technology. Pulse static, a primary cause of TV and radio disturbances, originates primarily from degraded insulation of transmission and distribution lines and thermostats in old refrigerators and freezers. However, this static occurs in an irregular, mainly intermittent manner, making identification of its origin difficult.

Researchers have solved this problem by developing a detector that measures instantaneous pulse static as well as continuous static waves and their direction.

Luminous multiporous silicon. Investigators are conducting basic research on this luminous element. The topics under study include the effects of the microscopic size of multiporous layers, the effects of its quantum size, and luminance caused by chemical compounds on the surface.

Virtual reality systems for the auditorium. Acoustic design of concert halls is often a hit-or-miss proposition. Its effectiveness would increase with a system that allowed the designer to hear the results of design changes. Researchers are developing a virtual reality system that would predict simply, conveniently, and quickly the acoustics of a hall by using reflective sound data reproduced by a computer based on design blueprints.

Acknowledgment

I am very grateful to Frank Nagashima for translating most of this material, which was made available to us only in Japanese.

Questions regarding this column can be addressed via e-mail to David K. Kahaner, US Office of Naval Research, Far East, at kahaner@cs.titech.ac.jp.

The PowerPC 601 Microprocessor

The PowerPC 601 microprocessor is the first of a family of processors based on the PowerPC architecture. The general-purpose processor contains a 32-Kbyte cache and a superscalar machine organization that allows dispatch and execution of up to three instructions each clock cycle. The bus interface and storage control mechanism can be configured for a wide range of system designs: low-cost desktop personal computers to high-performance multiprocessor systems.

Michael C. Becker

Michael S. Allen

Motorola, Somerset

Charles R. Moore

John S. Muhich

David P. Tuttle

IBM Austin, Somerset

The PowerPC 601 microprocessor is the first implementation of the PowerPC architecture developed by IBM Corp. and Motorola, Inc. at the Somerset facility in Austin, Texas. Together with Apple Computer, Inc., the alliance is developing a family of processors that addresses computer markets requiring high performance, low-power consumption, and outstanding price/performance.

The 601 microprocessor project achieved three significant goals. The first was to bring a PowerPC processor into the marketplace quickly. The alliance teams (referred to as "we" after this for brevity) achieved this by leveraging existing IBM and Motorola technology. IBM's POWER (Performance Optimized with Enhanced RISC) architecture[1] formed the basis of the PowerPC architecture, and the RISC Single Chip (RSC) processor[2] the base for the 601 microprocessor. Motorola contributed various architectural considerations that helped to form the PowerPC architecture. In addition, the bus interface from the Motorola 88110 microprocessor[3] served as a basis for the PowerPC 601 bus interface.

The second goal was to offer competitive performance at a low cost. We enhanced the base RSC design to allow more concurrent instruction execution and higher frequency operation. Integrating the processor onto a relatively small die and selecting an economical package minimized the cost.

The final goal was to offer capabilities suitable for a wide range of system design points. We achieved this by extending the versatile Motorola 88110 interface to support the PowerPC architecture, advanced protocol operations, and enhanced multiprocessing support.

Architecture overview

The PowerPC architecture is a third-generation RISC architecture designed for diverse computing requirements.[4] The result is a powerful architecture that embraces fundamental concepts of simplicity and general applicability. It is also extensible to advanced techniques that will allow it to be used in future generations of PowerPC microprocessors.

Architecture goals. We felt it important to maintain an application binary interface (ABI) that was compatible with IBM's POWER architecture. Such compatibility allows PowerPC-based machines to exploit the existing application base of the RISC System/6000 machines. To that end, we adopted the POWER user-level instruction set and programming model as a starting point for the architecture. Although some instructions were ultimately added and others deleted, these changes can be managed by compilers and operating systems.

0272-1732/93/1000-0054$03.00 © 1993 IEEE

We also wanted to simplify the architecture to ease unnecessary implementation constraints. This flexibility allows optimizations appropriate for specific market targets. In addition, the simplifications permit smaller, faster, and more aggressive superscalar implementations.

A third objective was to provide support for a wide range of uniprocessor and multiprocessor system configurations. Recognizing key abstractions of the storage hierarchy and defining the storage control architecture to allow effective management of these abstractions achieved this goal. The architecture also allows storage references to follow either a big-endian or a little-endian byte-ordering convention to support different operating system needs.

The final architecture goal was to define 64-bit extensions that allow upward compatibility of 32-bit applications. We defined 64-bit instruction operation and 64-bit memory management as a logical extension to the 32-bit execution model. To allow flexibility, each implementation can either comply with the base 32-bit PowerPC architecture or the extended 64-bit architecture.

Autonomous execution units. One fundamental concept of the architecture involves the partitioning of the architecture. The specification divides the execution of instructions into three logically distinct processing units: branch, fixed point, and floating point. The units are loosely coupled so instruction execution can occur concurrently. Note that this is an architectural partitioning that does not impose implementation constraints. For example, the architecture allows implementations to provide multiple copies of any of the units for added performance or to combine any of the units for more efficient silicon area use.

We structured the branch processor architecture to allow early handlings of branch instructions. Resources architecturally defined as part of the branch processor generate target addresses and evaluate branch conditions. This logical partitioning lets the branch processor completely remove branch instructions from the execution stream and execute them in parallel with operations occurring in the other functional units.

The branch processor architecture defines three user-accessible branch control registers and several forms of branch instructions. The link register in conjunction with certain branch instructions provides efficient subroutine linkage. The count register acts with conditional branch instructions to construct iterating loops. The condition register contains eight 4-bit condition fields, which are set by a wide range of instructions. Branch instructions can be conditional on a bit in the condition register, conditional on the state of the count register, conditional on both registers, or simply unconditional. The branch target address can be absolute, program counter relative, or indirect from either the link register or the count register. We also defined a set of instructions to allow logical operations and movement of fields in the condition register.

The PowerPC fixed-point architecture defines 32 general-purpose registers and a rich set of computational, logical, shift, and storage access instructions. A full complement of specified add, subtract, multiply, divide, and logical instructions operate on either 32-bit or 64-bit operands. In addition, the computational model allows the construction of extended-precision arithmetic operations from these base operations. The architecture also specifies a powerful set of rotate-with-mask and shift instructions.

The fixed-point processor controls addressing for all storage access instructions. Addresses are generated either from a base register, plus a displacement from the instruction, or from a base register, plus the value in an indexing register. These instructions allow byte, half-word, word, and double-word access to and from storage, and may also specify an automatic update of the base address register.

The architecture also specifies instructions for moving larger blocks of data between the registers and storage. The load/store multiple instructions are useful for subroutine linkage, and the load/store string instructions are useful for movement of byte strings.

The floating-point architecture defines 32 double-precision floating-point registers and a computational model that conforms to the IEEE Standard for Binary Floating-Point Arithmetic.[5] Computational instructions and storage access instructions support both single- and double-precision operands. In addition to the normal set of computational instructions, the architecture provides support for a powerful set of floating-point multiply-accumulate instructions.

The architecture allows implementation flexibility in the floating-point processor. Sophisticated implementations can commit the entire specification into hardware for high performance, while low-cost implementations could, for example, optimize the machine organization for efficient single-precision support. It is also possible for implementations to trap instructions and have software provide the required functionality.

Storage control. The architecture provides a robust storage control structure, which includes definitions for memory management, caching, and memory operations. The 32-bit architecture supports a 52-bit virtual address and a 32-bit real address. The 64-bit architecture supports an 80-bit virtual address and a 64-bit real address. Address translation occurs in two steps. A segmentation process translates the effective address into the virtual address, then a paging process translates the virtual address into a real address. Page translation is implemented through a memory-resident hashed page table. The page table is commonly cached in a translation look-aside buffer, so the architecture provides instructions to control the TLB. We fixed the page size at 4 Kbytes. In addition to the segmentation and paging mechanisms, the architecture also defines a block address translation mechanism that allows translation of blocks that can vary in size from 128 Kbytes to 256 Mbytes.

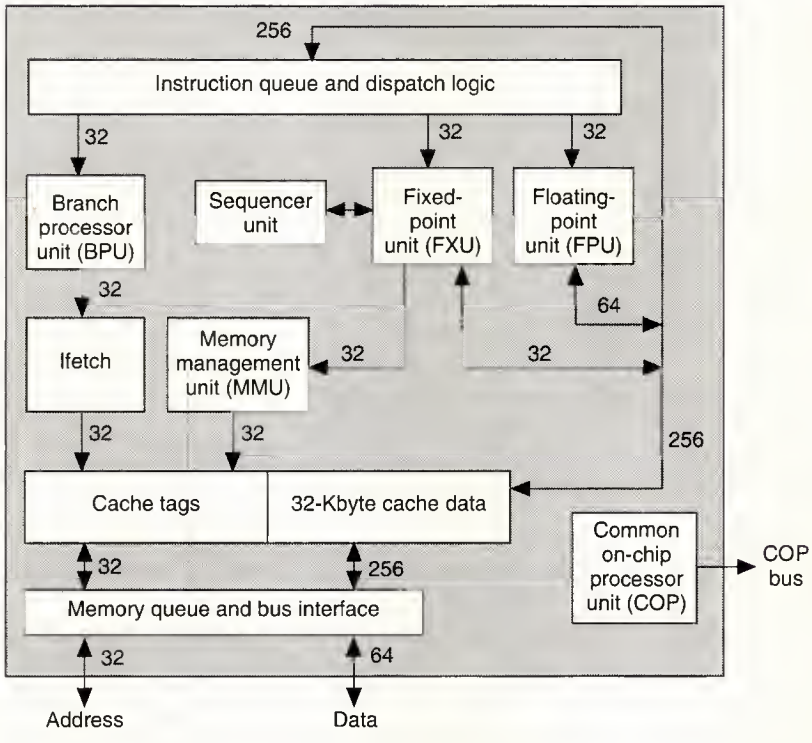


Figure 1. PowerPC 601 microprocessor block diagram.

The architecture specifies a generalized storage coherency model adaptable to a variety of coherency protocols. This model guarantees a certain behavior of storage references so multiple processors and devices can share memory. The specification gives implementations the freedom to define the particular protocols used to enforce coherency. A reservation mechanism coordinates with the coherency scheme to permit implementation of shared-memory synchronization primitives. Storage attributes are set at both page and block levels, and allow specification of cachable, write-through, memory coherent, and guarded storage. The architecture also defines cache control instructions to allow software-directed coherency and application tuning. In particular, it specifies instructions for data cache invalidate, instruction cache invalidate, data cache flush, data cache store, data cache allocate with zeroes, and data cache touch.

The PowerPC architecture defines a relaxed storage consistency model. This model guarantees sequential consistency as viewed by a program but allows considerable implementation flexibility in the ordering of memory references. This weak consistency model allows the processor to achieve higher levels of performance through speculative accesses, runtime reordering of storage accesses, and optimized use of the memory resource. The architecture also defines a set of synchronization and barrier instructions to allow software to achieve the effect of a stronger ordering model. These instructions are particularly useful for memory-mapped I/O applications.

The combination of the large address spaces, versatile memory management structures, and generalized storage coherency and consistency models will help to ensure the longevity of the architecture.

Machine organization

The PowerPC 601 microprocessor is a 32-bit implementation of the PowerPC architecture. The superscalar machine organization can dispatch up to three instructions each cycle. The 601 microprocessor also benefits from instruction pipelining, which helps to improve its clock rate. Figures 1 and 2 illustrate the block diagram of the 601 microprocessor, and Figure 3 shows the pipeline structures.[6]

The instruction queue and dispatch logic buffer instructions from the cache and dispatch up to three instructions each cycle—one each to the fixed-point unit (FXU), the floating-point unit (FPU), and the branch processing unit (BPU). The FXU communicates with the sequencer unit for handling infrequently used instructions and certain control-intensive tasks. In addition, the FXU interfaces with the memory management unit (MMU) for cache accesses. The 32-Kbyte, unified cache provides a 32-bit interface to the FXU, a 64-bit interface to the FPU, and a 256-bit interface to both the instruction queue and the memory queue. The chip interface includes a 32-bit address bus and a 64-bit data bus. In addition, the chip supports an asynchronous serial port, the common on-chip processor (COP) bus, to support debugging and test features.

Instruction queue and dispatch unit. The 601 microprocessor contains an eight-entry instruction queue for prefetched instructions. The queue is fed by an eight-word bus from the cache. During each cycle, the dispatch logic considers the bottom four entries of the instruction queue and dispatches up to three instructions (Figure 4). The queue supports the full range of possible shift amounts with no instruction alignment restrictions on the program.

The queue positions from which instructions can be dispatched vary for each function unit. Branch instructions and most floating-point instructions can be dispatched from any of the bottom four entries of the instruction queue. Fixed-point instructions are always dispatched from the bottom queue entry. Floating-point stores are dispatched to both the

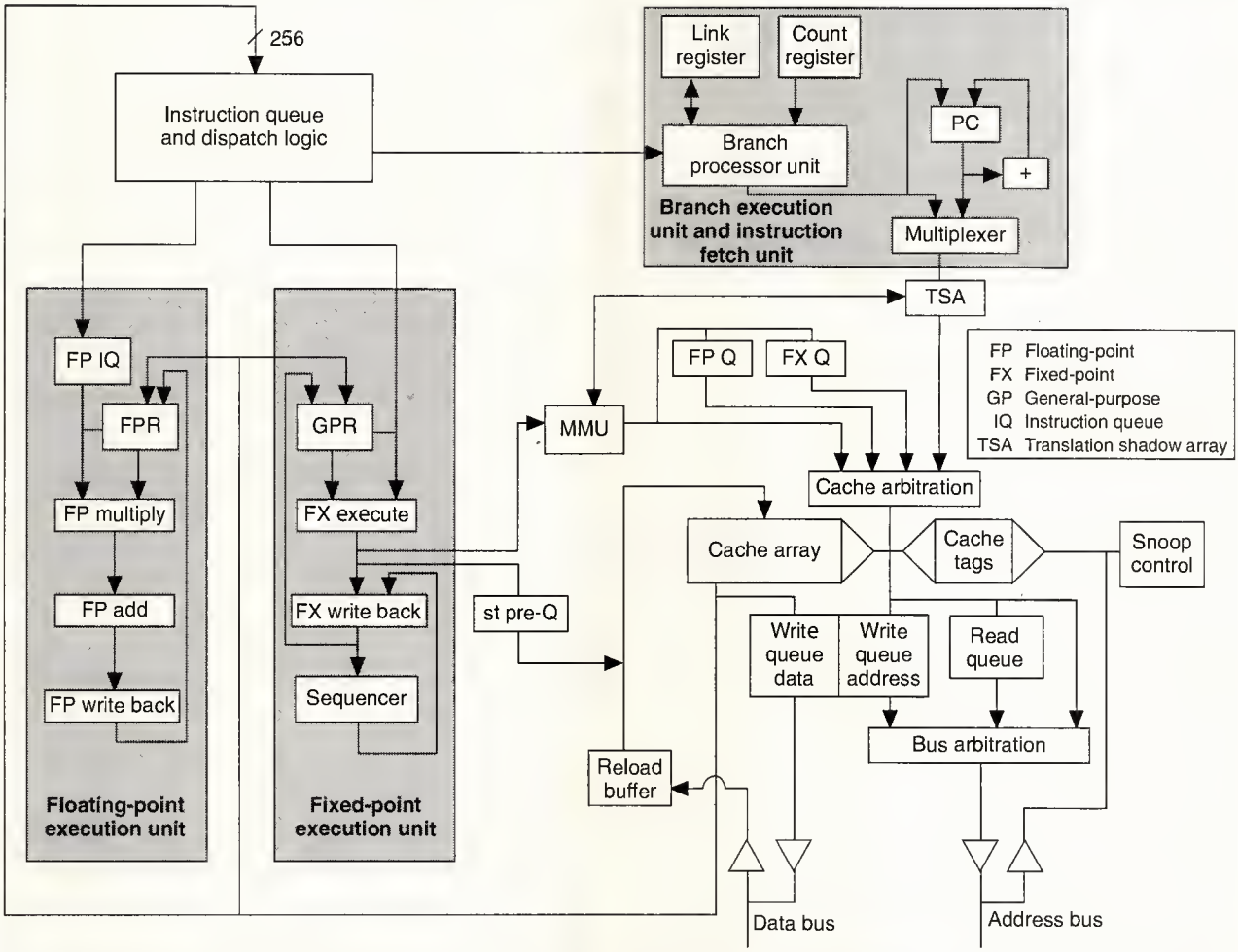


Figure 2. Internal dataflow diagram.

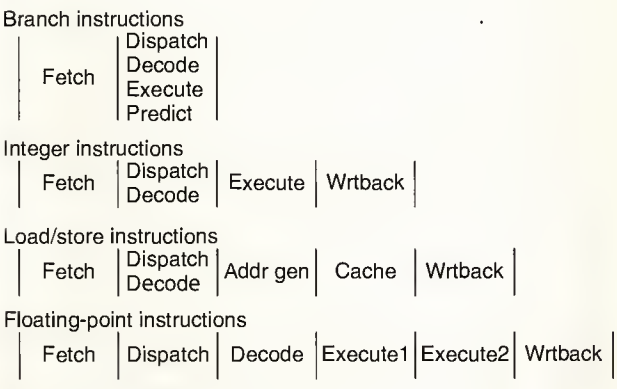


Figure 3. Pipeline description.

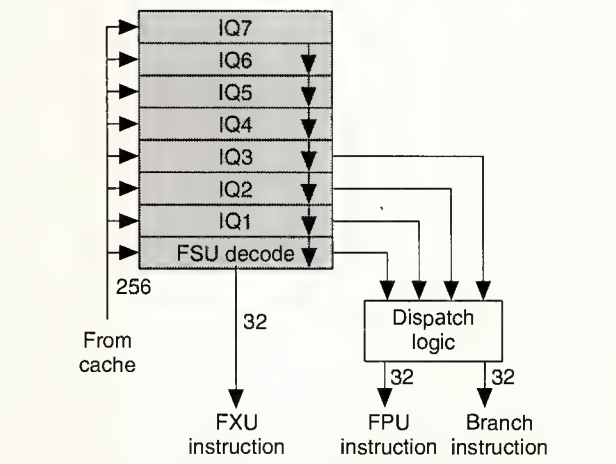


Figure 4. Instruction queue and dispatch logic.

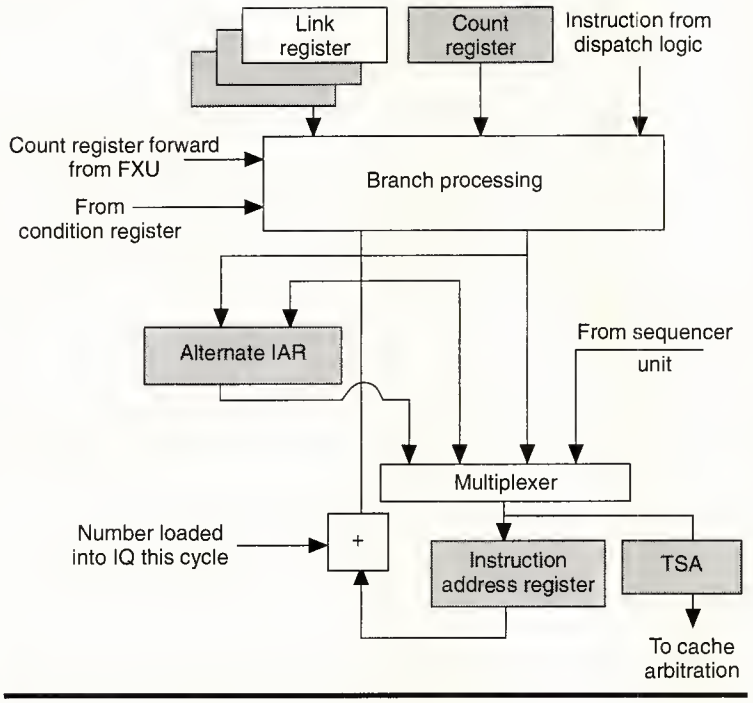


Figure 5. Instruction fetch and branch processor units.

FXU (for address generation) and the FPU (for data sourcing). Floating-point loads are dispatched only to the FXU, but the FPU is made aware of any dependencies, and data transfers directly to the FPU.

The 601 microprocessor permits out-of-order dispatch such that branch instructions and most floating-point instructions can be dispatched and removed from the queue even if a preceding fixed-point instruction is interlocked. These folded instructions execute concurrently and do not occupy a position in the fixed-point pipeline. A tagging and counting mechanism preserves the program order completion of these instructions. Out-of-order dispatch is more complex to implement, but it allows the 601 microprocessor to expose subsequent branches earlier, which can reduce the dispatch stalls that may otherwise result.

Instruction fetch unit. This unit coordinates instruction fetching from the cache. Several different sources in the 601 microprocessor generate instruction fetch addresses. The branch processor provides the address that results from branch instructions. The sequencer unit provides addresses associated with interrupts and other synchronizing events. The instruction fetch unit itself generates the next sequential address in the event that no branch or interrupt has occurred. During each cycle, the appropriate address is selected, translated, and forwarded to the cache arbitration logic for consideration to access the cache. The instruction queue and dispatch logic accepts and processes instructions fetched from the cache.

The instruction fetcher also provides fast address translation of instruction fetch addresses through the translation shadow array. The TSA automatically keeps track of the four most recently used instruction address translations and provides fully associative comparison in response to the address generation of any instruction fetch. The TSA supports both page- and block-oriented address translations. In the event of a miss in the TSA, the instruction fetcher arbitrates for access to the 601's primary MMU for translation, and then updates the TSA based on a least recently used replacement (LRU) algorithm.

Branch processor unit. The BPU (Figure 5) executes branch instructions in its two-stage pipeline (Figure 3). The 601 microprocessor dispatches, decodes, evaluates, and, if necessary, predicts the direction of branch instructions in one cycle. On the next cycle, the resulting fetch can access new instructions from the cache. This allows the processor to quickly react to branches detected in the instruction stream and to reduce the latency of subsequent instructions.

Unconditional and count register dependent branches execute during the same cycle that they are dispatched. As a result, they present no delay in the instruction dispatch stream to the FXU or the FPU and effectively execute in zero cycles. Conditional branches dependent on the condition register can either be resolved or unresolved at the time of dispatch. Each of the eight fields of the condition register has a set of associated interlocks activated by instructions that update that field. If a conditional branch is dependent upon a noninterlocked condition register field, the branch can be resolved immediately. In this case, the condition is evaluated based on the contents of the condition register, and the branch presents no delay in the instruction dispatch stream to the FXU or FPU.

Branches conditional on an interlocked condition register field are unresolved. In these cases, the 601 microprocessor employs a static branch prediction algorithm to predict the direction of the unresolved branch. The 601 microprocessor algorithm predicts that the branch will be taken if the displacement of the target address is negative, and predicts it will not be taken if it is positive. As an aid for compiler-directed prediction, a bit in the branch instruction opcode allows this prediction scheme to be reversed. Instructions fetched on behalf of a predicted conditional branch can be conditionally dispatched, but they do not execute until the prediction has been validated.

Eventually, condition register interlocks are released, and the validity of the prediction can be checked. If the prediction was correct, the branch completes and all prefetched instructions are marked eligible for execution. In the event that the prediction was incorrect, any incorrectly prefetched instructions are discarded and fetching resumes down the correct path.

The 601 microprocessor has several features to speed up recovery after a misprediction. First, a fast alternate address restore mechanism saves the alternate path address and immediately redirects the instruction fetcher. Second, the FXU forwards the results of compare instructions directly to the branch processor as well as the condition register. This allows earlier resolution of unresolved conditions. Finally, the instruction queue implements a delayed purge mechanism, which retains any prefetched sequential instructions beyond the conditional branch until the prefetched predicted instructions are made available for loading into the instruction queue. In the event that the condition is resolved before the predicted instructions are available (the likelihood of which is increased by the optimizations for the compare instructions in the FXU), and the prefetch was incorrectly predicted down the taken path, dispatch simply continues down the sequential stream. This stream consists of the instructions remaining in the instruction queue.

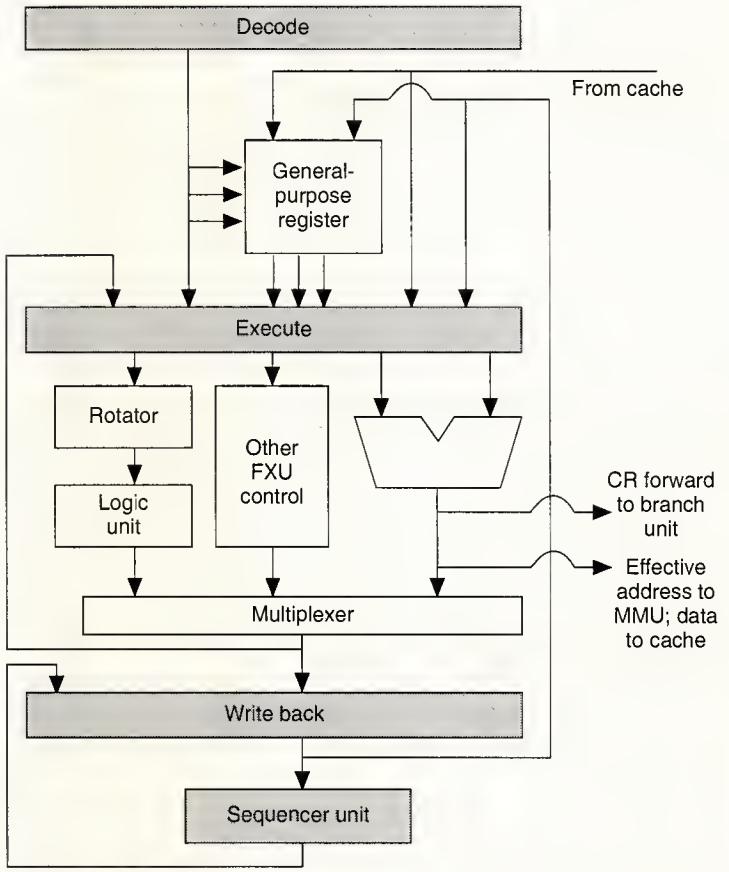


Figure 6. Fixed-point execution unit.

Fixed-point execution unit. The FXU receives instructions from the instruction dispatcher. In general, the FXU serves as the master pipeline, managing the synchronization control required to achieve precise exceptions. The instructions execute in a four-stage pipeline (fetch, decode, execute, write back; Figure 3), and the hardware interlocks all hazards (Figure 6). All architecture instructions process in hardware, and most execute in a fully pipelined manner. Some of the arithmetic (multiply and divide) and multiple word storage operations require several cycles in the execute stage. To improve the effective execution latency, the FXU forwards any register-dependent results to their respective function units at the same time the data is written back to the register file. In addition, although the differences between the POWER architecture and the PowerPC architecture at the user level are small, the 601 microprocessor also provides hardware support for all POWER architecture user mode instructions.

The FXU handles the address generation portion of all load and store instructions. This includes the floating-point loads and stores, although in some cases, the instructions also progress through the floating-point pipeline. A dependent operation following a load instruction causes a one-cycle load delay slot. The FXU sequences the execution of the load/store multiple instructions and the load/store string instructions. The hardware handles most forms of unaligned addresses; however, in some cases, multiple accesses from the cache are necessary.

The FXU features a fast execution of compare instructions and the ability to forward results directly to the branch processing unit. As discussed previously, this allows efficient handling of branches dependent on the result of the compare instruction.

Floating-point execution unit. The FPU (Figure 7, next page) complies with the IEEE-754 standard for both single- and double-precision floating-point arithmetic operations. The FPU supports the compound multiply-add operations that are defined in the PowerPC architecture. Furthermore, the hardware automatically handles all floating-point data types including denormalized numbers, NANs, QNANs, and infinities. The FPU pipeline has six stages (Figure 3). The decode stage contains 32 double-word registers, the instruction de-

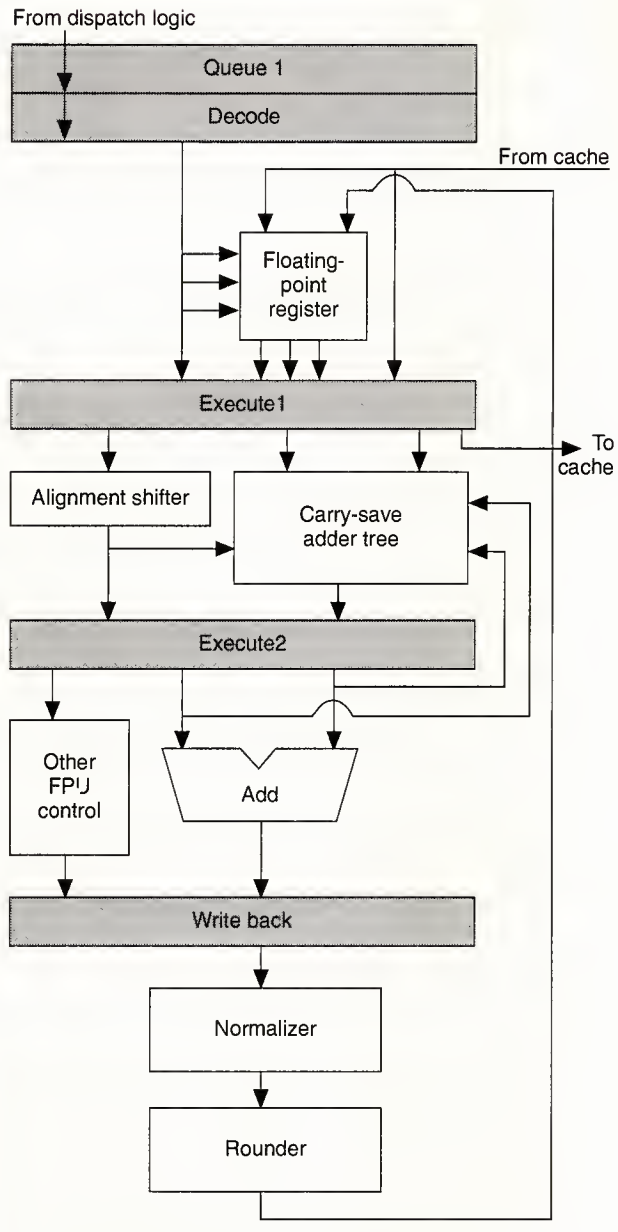


Figure 7. Floating-point execution unit.

code logic, and the main pipeline control for the FPU. The execute1 (or multiply) stage contains the Booth encoder and carry-save adder tree, and an alignment shifter for the second operand. The execute2 (or add) stage accepts the sum and carry values from the previous stage, and produces a single intermediate result. Finally, the write-back stage rounds and normalizes results, and updates the register.

The FPU is optimized so that most single-precision operations (all except divide) and most double-precision operations (not involving multiplication or division) that involve operands with normal data types are fully pipelined. This achieves a four-cycle latency and one-cycle throughput. Double-precision operations involving multiplication are obtained by double pumping each pipeline stage to achieve a five-cycle latency and a two-cycle throughput (Figure 3). Division is performed using a 2-bit nonrestoring division algorithm, which takes 17 cycles for single precision and 31 cycles for double precision.

The FPU receives instructions from the instruction dispatch unit. In many floating-point-intensive applications, it is possible that the FPU will lag behind the full dispatch rate of the dispatch unit. To alleviate the adverse effects of this, the FPU maintains an additional two-entry instruction queue. This allows the dispatch logic to remove floating-point instructions from the primary instruction queue and expose subsequent fixed-point and branch instructions earlier.

The FPU operates independently from the FXU and can concurrently execute floating-point instructions. A carefully tuned synchronization scheme allows these execution units to progress independently while maintaining precise exceptions. These pipelines cooperate in the execution of floating-point storage access instructions but are not required to proceed in lock step. For example, the FXU can process the addressing portion of a floating-point store operation and pass it onto the cache unit without waiting for the FPU to produce the store data. The cache unit will complete the store operation once the FPU provides the data.

The 601 microprocessor provides a mode that forces enabled floating-point exceptions to be reported in a precise manner. This useful mode also serves as a software debugging aid for floating-point programs that produce degenerate results.

Sequencer unit. This unit is essentially an embedded support processor that assists the core CPU in handling many of the algorithmic functions of the PowerPC architecture. It sequences the power on reset functions, which include array self-testing for the cache and array initialization for all other storage cells on the chip. It maintains an on-chip real-time clock and the system decrementer function. In addition, it handles the recording and sequencing of interrupts, context-synchronizing events, and errors.

The sequencer also walks through the hashed page table for address translations that miss in the TLB, and, as appropriate, it updates the storage access recording bits. In addition, to minimize the chip area overhead, several of the less frequently used system control registers are physically implemented in a RAM structure within the sequencer. As a result, the instructions that operate with these registers are passed from the FXU to the sequencer for execution assist.

Memory management unit. The MMU (Figure 8) per-

forms the virtual-to-real address translation for load and store instructions. In addition, it acts as a backup for instruction fetch address translations that miss in the instruction fetcher's TSA. The MMU is tightly coupled to the FXU so that address translation can occur during the execute phase of the FXU pipeline. A cache access occurs the following cycle using the translated address. Arbitration logic determines whether the FXU or the instruction fetcher has use of the MMU during a particular cycle. The instruction fetcher's use of the MMU should be infrequent since the hit rate of the TSA is high for most applications.

The MMU provides support for segment, page, and block translations. Address translation begins with indexing into one of 16 segment registers. Control bits in the segment register control the mapping of an address to the real address space, the I/O space, or the virtual space. Virtual addresses are further translated by page or block translations mechanisms.

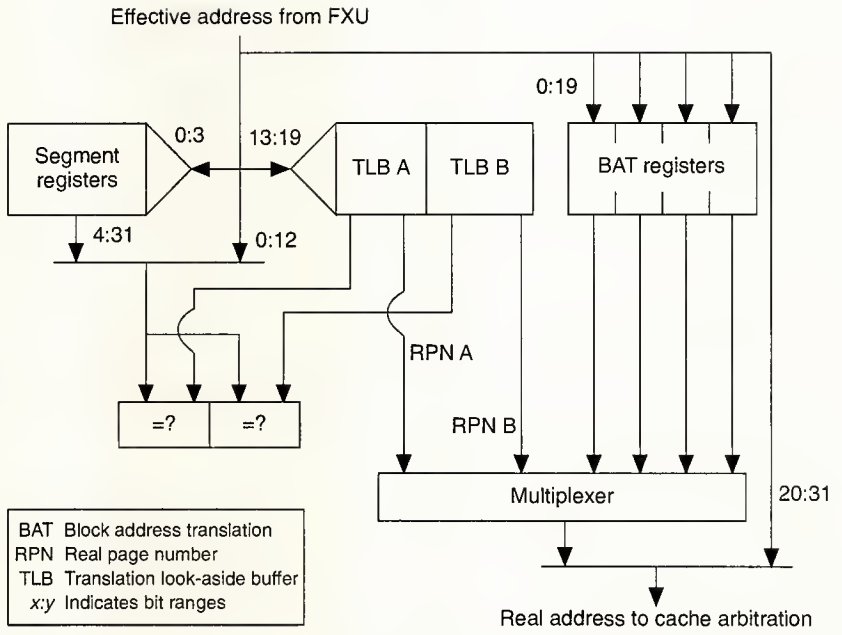


Figure 8. Memory management unit.

The PowerPC architecture specifies a hashed page table structure and a fixed-page size of 4 Kbytes. Portions of the virtual address are used to hash into the page table entries, which are subsequently examined for matching virtual page numbers. A matching entry produces a real page number, which can then be used to access real memory. If no match is found after 16 attempts, a page fault exception is taken. In an effort to speed up the page-based address translation, the 601 microprocessor provides a 256-entry, two-way set-associative TLB cache that is used for both instructions and data. Updates to the TLB and maintenance of the storage access recording (reference and change) bits are handled in hardware.

The BAT, or block address translation, mechanism allows for up to four block mappings that can vary in size from 128 Kbytes to 8 Mbytes. This is useful for handling large sections of memory that are not subject to demand paging but still require address translation and protection. Proper use of the block translation registers can reduce TLB thrashing and increase overall processor performance.

Cache structure. The 601 microprocessor provides a unified 32-Kbyte, eight-way set-associative cache (Figure 9, next page). Each 64-byte line holds two 32-byte sectors; cache indexing and cache tag comparisons use the real (physical) address. In general, the cache makes use of a store-in (or copy-back) policy to store data. However, the cache allows page- and block-level control of cachability, write through, and coherency (via the MMU). An LRU cache replacement algorithm is used, and coherency is maintained on a sector basis using the four-state MESI (modified, exclusive, shared, invalid) cache coherency protocol.[7]

In a unified cache implementation, it is important to make the most effective use of the available cache bandwidth. The 601 microprocessor employs several techniques to achieve this. First, the cache interface to the fetch unit and the memory queue is 8 words wide. At 66 MHz, the data rate for this interface is 2.1 Gbytes/s. Although all eight words are not used every cycle, this bandwidth speeds sector cast-outs (cast-outs refer to modified data moves to memory), snoop pushes, and instruction fetching. Second, the cache is capable of a complete read-modify-write every cycle, which allows the 601 microprocessor to process store operations in one cache cycle. Third, a balanced arbitration scheme prioritizes the various cache access requests that can occur each cycle. To prevent cache arbitration from stalling the execution units, operation queueing is provided above the cache for fixed-point and floating-point storage operations. Finally, the cache is a nonblocking design, and a memory queue sits below the cache to hold pending storage operations.

The 601 microprocessor cache supports all PowerPC cache control operations. These are useful for tuning applications in which memory performance can be improved by explicit cache management, and for systems that perform software-assisted coherency schemes.

The cache is a full-custom design that exploits leading-

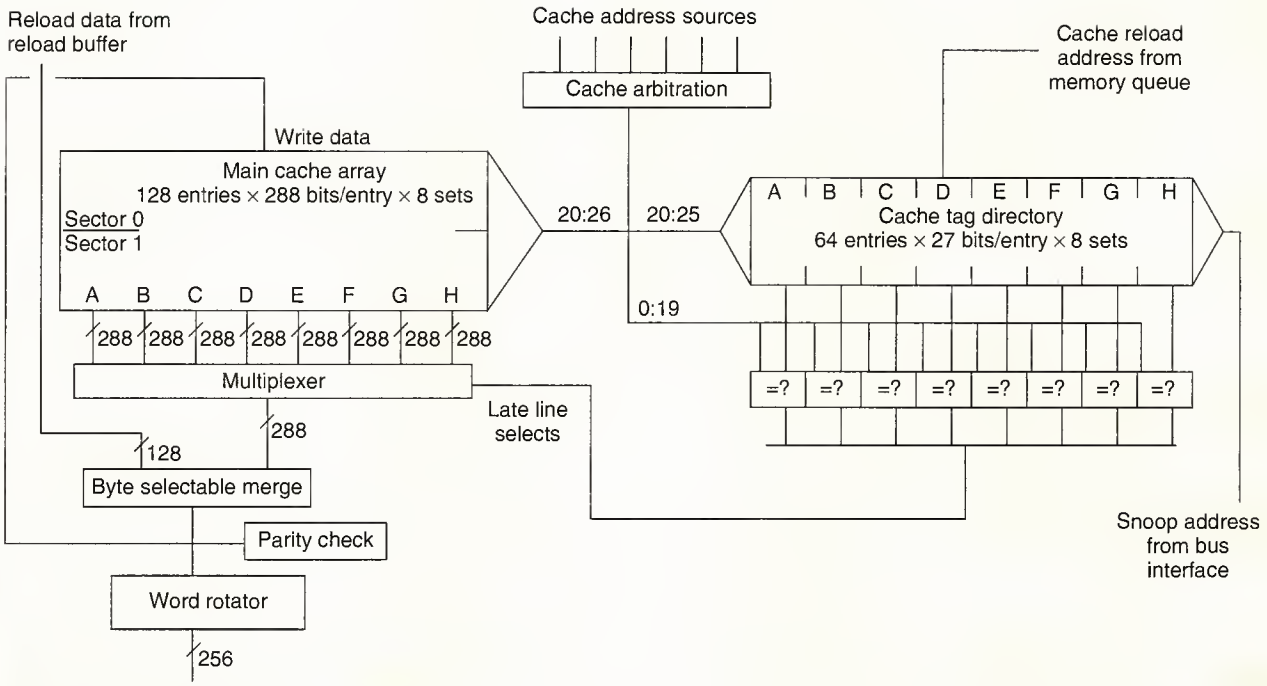


Figure 9. Cache organization.

edge circuit design techniques to achieve the required density and performance characteristics. The cache uses an electrically stable six-transistor static RAM cell and is protected with byte parity for enhanced error detection and reliability. A deterministic self-test procedure checks the cache as part of the power-on reset process.

Memory queue. The memory queue cooperates with the cache and the bus interface unit to queue memory transactions. It decouples the cache from the bus interface, which allows the cache to continue providing data for instruction fetches and load/store operations, independently of the bus transactions still in the queue.

The memory queue is made up of a two-entry read-address queue and a three-entry write-address/data queue. The write queue holds up to three 32-byte sectors of dirty (modified) data. The read queue, the write queue, and in some cases the cache itself, arbitrate for access to the 601 bus. The memory queue maintains full hardware-enforced coherency to the processor, cache, and memory. Additionally, the memory queue supports a dynamic reload feature, which permits background fetches for the other sector of a cache line after the critical sector has been successfully loaded. When enabled, this feature provides an automated cache prefetching capability for both instructions and data.

Bus interface unit. The 601 BIU translates memory requests from the memory queue and the cache into transactions on the bus interface. It has a 32-bit address bus and a 64-bit data bus and uses an efficient bursting protocol to achieve high data throughput. The TTL- or CMOS-compatible interface runs at the processor clock rate or any integer multiple of the processor clock period. Figure 10 is a block diagram of the BIU with the memory queue.

The BIU employs a bursting protocol that uses one address to transfer four beats of data (8 bytes each). While the theoretical limit of data bus throughput is the product of bus width and bus frequency, the realizable bandwidth is limited by factors such as interface technology, bus arbitration overhead, address transfer overhead, memory latency, and interprocessor communication. In the 601 microprocessor, consecutive burst transfers require one-cycle separation for an effective throughput of 422 Mbytes/s at 66 MHz. The protocol specifies independent operation of the address and data buses to allow the bus to approach its bandwidth limit.

A simple bus protocol employs a tenured bus in which the memory latency governs the use of the address, as shown in the first example in Figure 11. If the memory latency is long, the actual data bus bandwidth can be significantly degraded. To overcome this problem, the 601 address and data buses operate independently, and new addresses can be launched before the preceding data transfer has completed, as shown in the address pipelining example. The 601 microprocessor itself can pipeline up to two outstanding addresses on the bus. In multiprocessing

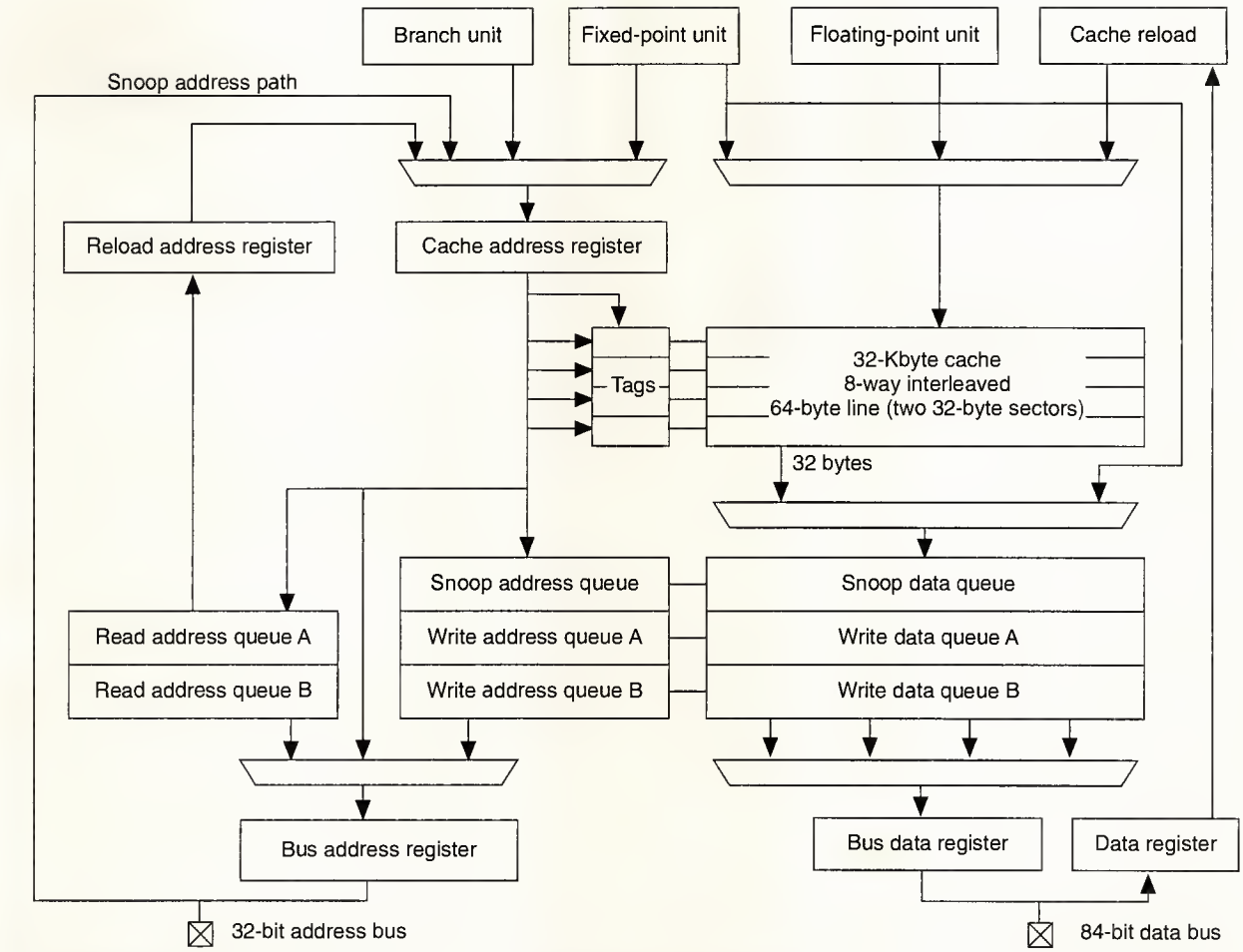


Figure 10. Memory path block diagram.

configurations, the protocol can support out-of-order completion of transactions and a greater number of outstanding addresses. This further improves bus efficiency in systems with variable memory latency (system I/O vs. memory, cache memory, banked memory designs).[8]

The interface protocol uses three phases to control the address and data buses. Figure 12 (next page) is a diagram of a basic 601 bus transaction illustrating the three phases. The following outlines the basic operation of each of the phases defined for the protocol:

- **Arbitration.** Arbitration signals request and grant ownership separately for the address bus and the data bus.
- **Transfer.** The address bus and the transfer

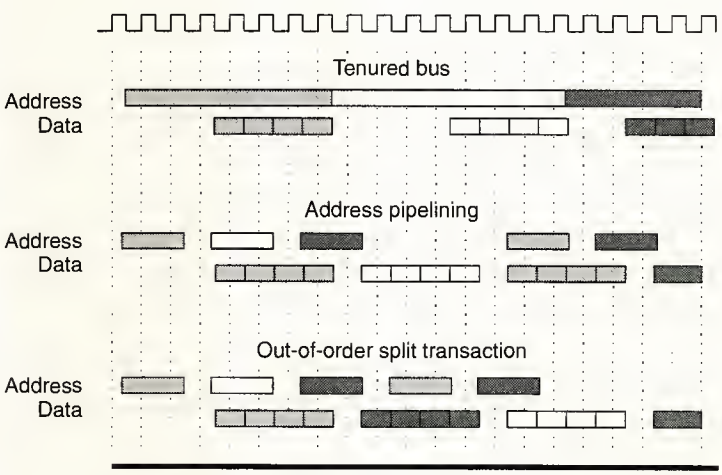


Figure 11. Bus optimizations.

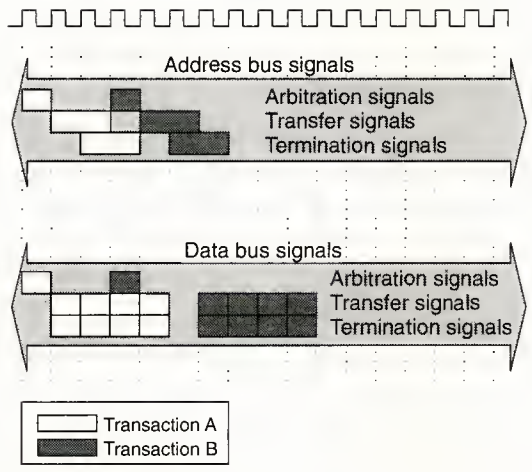


Figure 12. Bus transfer protocol.

attribute signals are driven to designate the location, type, and size of the transfer. Transfers include burst, single-beat, reservation, broadcast cache control, and broadcast TLB control operations. Either the 601 microprocessor or the target of the transfer drives the data bus signals.

- **Termination.** Termination signals conclude the bus tenure separately for each bus.

Reservation transfers are bus operations initiated by the reservation mechanism and are used to achieve shared-memory synchronization. The program-initiated cache and TLB broadcast operations provide remote control of caches and TLBs in other processors on the bus. These address-only transactions are illustrated in the middle of the address pipelining example (Figure 11). Since the data bus is typically more heavily used than the address bus, these address-only operations can transparently use the spare bandwidth of the address bus to data bus operations.

The 601 microprocessor maximizes opportunities to use the bus resource by prioritization of internal requests for access to the bus. As discussed earlier, the PowerPC architecture defines a weak consistency model, which, for most instruction streams, allows memory operations to complete in a different order than programmed. This has two positive effects on bus throughput. By allowing the longer latency read operations to start on the bus before the older (with respect to program order) store operations, the hardware reduces the latency of the read operation. Also, placing the dirty cache sector cast-outs and the dynamic reload requests lowest in priority allows these opportunistic operations to attempt to fill the gaps between higher priority bus operations.

The 601 microprocessor minimizes memory access latency through bus parking, speculative forwarding of data, and a short cache-to-memory path. Bus parking eliminates an arbitration cycle by allowing the external arbiter to grant the 601 microprocessor the bus before the 601 makes a request. Speculative data forwarding is permitted through the definition of a data retry function that can be used to invalidate the double word transferred on the previous bus clock cycle. This is useful for systems employing secondary cache systems that forward data prior to the tag comparison and for correction algorithms that protect memory integrity. The 601 microprocessor reflects substantial design effort to minimize latency from cache miss detection to the return of instructions or data to the cache.

Bus timing example. System-level optimizations are critical to performance, particularly as the processor core often operates at a multiple of the bus clock frequency. Reducing memory latency at the system bus level by a bus clock cycle can potentially reduce latency as seen by the processor by several clock cycles. The 601's innovations to reduce latency are best shown by a best case example using the following assumptions:

- the bus is running at the same frequency as the processor;
- there are no higher priority operations queued;
- the 601 microprocessor is parked on the bus and does not need to assert a request; and
- the 601 microprocessor is connected to a fast, direct-mapped second-level cache using the speculative data-forwarding capability of the 601 interface.

In clock cycle 1, the physical address is driven to the cache directory, the two read queues, and the bus address register in parallel. One of the read queues is loaded, as well as the bus address register. Also in cycle 1, the result of the cache directory indicates a cache miss; therefore, the bus controller can drive the address to the bus on the following cycle. Since it has already been granted the address bus, the 601 microprocessor drives the address in clock cycle 2. (Figure 13).

The memory system responds with the first double word of data in clock cycle 3, and the subsequent double words of data in clock cycles 4, 5, and 6. The bus interface logic gathers two beats (four words) of data together before requesting access to the cache in clock cycles 5 and 7. The write back into the cache takes place late in clock cycle 6. The requested datum or instruction packet is simultaneously forwarded to the FXU, FPU, or branch unit in clock cycle 6.

Deadlock prevention in hierarchical buses. A special scenario arises in hierarchical bus environments that consist of a processor bus connected via a bridge component to an I/O bus (like EISA). If the processor and a master on the I/O bus simultaneously request data from the other bus, one of the requests cannot continue. A processor may have its request retried so that the I/O bus request can proceed (available on the 601 microprocessor). However, this solution may cause

difficulties with devices that should see an address only once for a single request. The 601 microprocessor overcomes this by allowing a processor request to be suspended after the address has been transferred, thus allowing the I/O bus to proceed and complete before the 601's transfer is completed.

Multiprocessing features. The 601 microprocessor maintains coherency with a granularity of 32 bytes (equal to a cache sector). Granularity refers to the smallest block of memory that is acted on to maintain coherency. The 601 microprocessor also distinguishes between global and nonglobal transactions. Only global transactions are monitored (snooped) by other processors.

The bus supports the MESI coherency protocol used by the cache. Each bus transaction is classified, so the other processors maintain coherency. Other processors use this information to update the MESI state of their cache entries. Conversely, the 601 microprocessor must snoop the bus for addresses in its own cache and generate an appropriate snoop response (shared hit, dirty hit, miss).

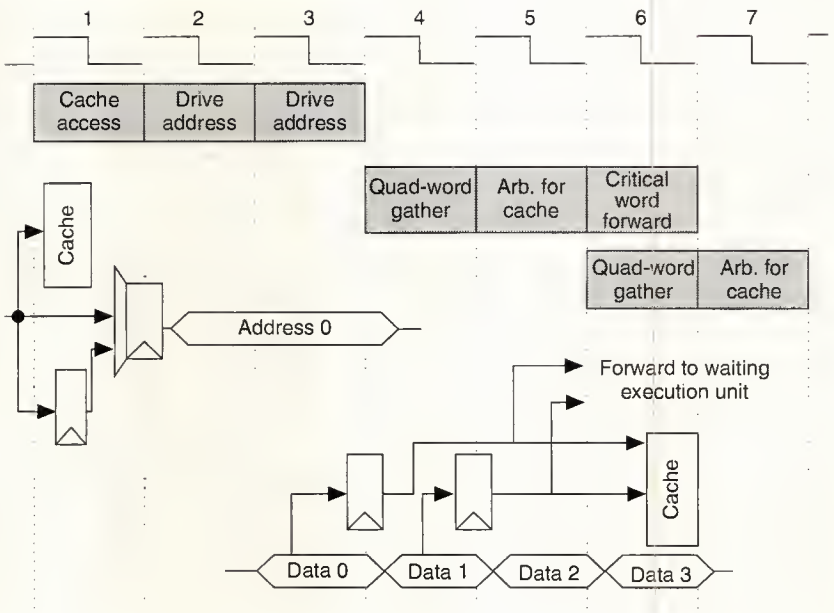


Figure 13. Cache miss timing.

A critical feature of the overall bus performance in multiprocessor systems is the snoop response latency, defined as the time the snooper takes to respond to a valid address on the bus with a coherency reply. The 601 microprocessor has only two bus clocks of snoop response latency because of its dual-ported cache tags. Since this is uniform across all 601s in a multiprocessor system, the latency associated with the hardware-enforced coherency is deterministic and very short.

Sample 601 microprocessor system configurations. The 601's bus interface accommodates a wide range of system configurations. As an overview of the 601's bus capabilities, we show three systems in Figure 14 (next page). Figure 14a is a block diagram of a 601-based single-processor system. In this system, a simple arbiter can determine whether the memory bus is owned by the 601 microprocessor or by an expansion bridge to a system bus such as VME, NuBus, MicroChannel, and PCI [9].

In Figure 14b, the 601 microprocessor interfaces to a second-level cache to provide improved uniprocessor performance. The 601 microprocessor provides a full complement of cache control signals to allow construction of a low-latency, high-performance cache. In this class of system the frame buffer will often attach to the memory bus for high performance.

The system in Figure 14c shows a multiprocessing system. Typical features of a 601-based multiprocessor might include a shared memory, a shared bus to facilitate hardware enforced coherency among a number of tightly coupled processors, a local cache system for improved processor and system performance, and cache control operations broadcast on the bus to allow processors and external hardware to control the local cache state. System designs using other interconnection topologies (data crossbars, hierarchical caches) can also be constructed.

Support for debugging and test

The common on-chip processor (COP) is the master control logic for the built-in self-test, debugging, and test features of the 601 chip. A simple serial command interface allows external devices to communicate with COP and initiate various actions. This command interface can be configured to be compatible with the IEEE standard[10] for JTAG access to boundary scan. The COP features combine to create a powerful debugging environment with detailed visibility into the 601 microprocessor.

Chip and packaging technology

We implemented the 601 microprocessor in an IBM CMOS technology with 0.6-µm minimum feature size and four levels of metal wiring. The 601 microprocessor is packaged in a 304-pin ceramic quad flat pack that is bonded using IBM's C4 solder ball technology. Table 1 summarizes other key physical characteristics of the 601 microprocessor.

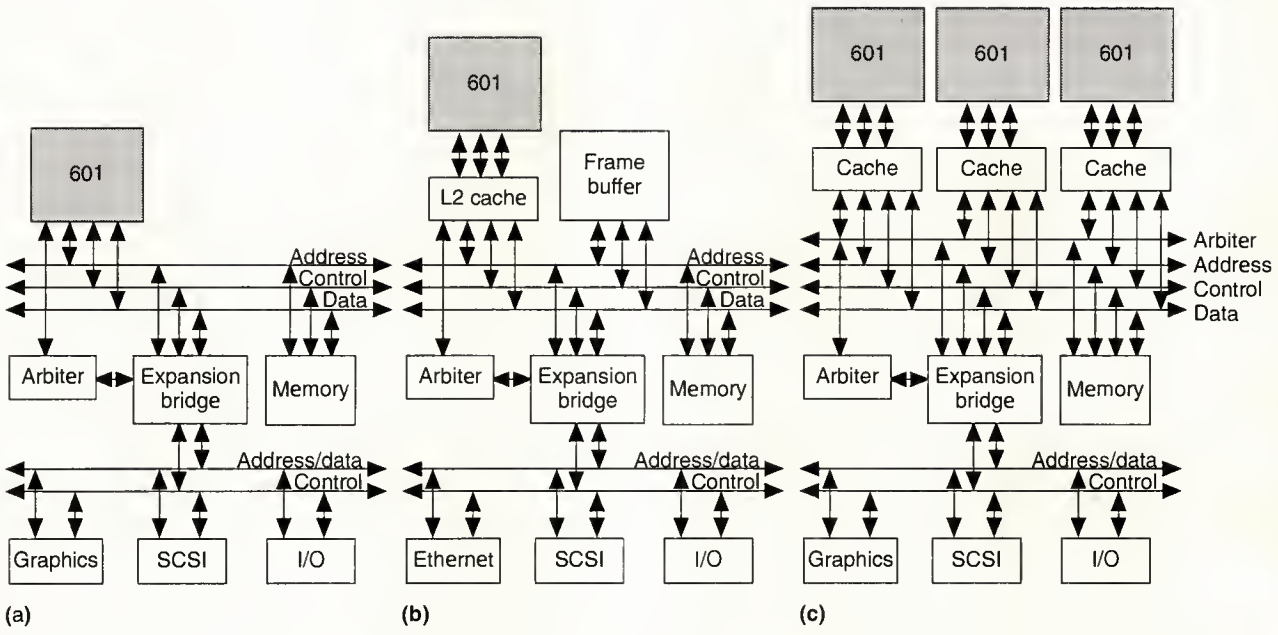


Figure 14. Typical 601-based systems: single processor (a), uniprocessor plus second-level cache (b), and multiprocessing system (c).

Table 1. Physical characteristics of the PowerPC 601 microprocessor.

Description	Characteristic
Technology	0.6-µm CMOS; four metal levels
Die size	10.95×10.95 mm
No. of transistors	2.8 million
Voltage	3.6
Power	6.5W at 50 MHz (typical)
Package	304-pin ceramic quad flat pack
Frequency	50 MHz and 66 MHz, initially
I/Os	184 signals; CMOS or TTL levels compatible

Table 2. Power PC 601 microprocessor performance.

Frequency	SPECint92	SPECfp92
66 MHz	60*	80*

*Estimated performance

Performance

In addition to the key performance features of the 601 microprocessor, other factors, such as the operating system, memory characteristics, system support, and operating frequency, play an important role in the overall performance of the system. To establish a rough basis for comparison, Table 2 provides estimated SPEC performance for a hypothetical system with a 66-MHz PowerPC 601 processor. Note that these are estimated projections and do not reflect the performance of a particular system.

THE DEVELOPMENT OF THE POWERPC 601 microprocessor was a unique project. It had aggressive functionality, performance, die size, quality, and schedule goals. It required the total dedication of many people from several divisions within IBM, from the Motorola staff, and from the Apple staff. The quick development of a close working relationship of these different groups is a testament to the dedication of the teams. This dedication is also reflected in the success of the 601 microprocessor.

The PowerPC 601 microprocessor successfully combines the architectural advantages of the PowerPC architecture, an efficient superscalar machine organization, and state-of-the-art CMOS technology to achieve an impressive level of functionality and performance. Critical performance features in the 601's machine organization include the branch unit, which can remove branches from the instruction stream, and the low-latency, high-density cache. In addition, the bus interface provides a flexible interface and a proven multiprocessor coherency protocol.

The 601's small die size and low-power consumption set new standards for microprocessors in its performance class. In the future the alliance will produce three processors that address low power, improved performance cost, and high-end performance. The combined efforts of IBM, Motorola, and Apple on the 601 microprocessor establish a firm foundation for the PowerPC family.

Acknowledgments

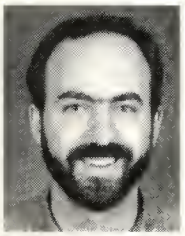
The authors acknowledge the contributions of the PowerPC Architecture Committee, logical designers, physical designers, verification and test engineers, packaging team, methodology support engineers, and all the other support staff who worked so hard and made many personal sacrifices. The result is a credit to their competence and dedication.

References

1. R.R. Oehler and R.D. Groves, "IBM RISC System/6000 Processor Architecture," *IBM J. Research and Development*, Vol. 34, No. 1, Jan. 1990, pp. 23-36.
2. C.R. Moore et al., "IBM Single-Chip RISC Processor (RSC)," *Proc. Int'l Conf. Computer Design*, Oct. 1992.
3. K. Diefendorff and M. Allen, "Organization of the Motorola 88110 Superscalar RISC Microprocessor," *IEEE Micro*, Vol. 12, No. 2, Apr. 1992, pp. 40-63.
4. E. Silha and G. Paap, "PowerPC: A Performance Architecture," *Proc. Compcon*, Feb. 1993, pp. 104-108.
5. *ANSI/IEEE 754-1985 Standard for Binary Floating-Point Arithmetic*, IEEE, Piscataway, N.J., 1985.
6. C.R. Moore, "The PowerPC 601 Microprocessor," *Proc. Compcon*, Feb. 1993, pp. 109-116.
7. *Future+, Logical Layer Specification, Draft 8.02 P896.1/D8.02*, IEEE Computer Society Press, Los Alamitos, Calif., Sept. 1989.
8. M. Allen and M. Becker, "Multiprocessing Aspects of the PowerPC 601 Microprocessor," *Proc. Compcon*, Feb. 1993, pp. 117-126.
9. *PCI Local Bus Specification, Revision 2.0*, PCI Special Interest Group, Hillsboro, Ore., Apr. 1993.
10. *IEEE Std 1149.1 Test Access Port and Boundary-Scan Architecture*, IEEE, Piscataway, N.J., 1990.

Charles R. Moore, an advisory engineer in the Advanced Workstation and Systems Division (AWS) of the IBM Corporation, is currently assigned to Somerset's High End Processor Development Group. He is the co-lead architect of the first PowerPC microprocessor, the 601. Previously, he was one of the key designers of the branch unit of the original RISC System/6000 chip set and was also heavily involved in the development of the RISC Single Chip (RSC) Power microprocessor.

Moore received a BSEE degree from Rensselaer Polytechnic Institute and an MSEE from the University of Texas at Austin. He is a member of the Institute of Electrical and Electronics Engineers and the Computer Society.



Michael C. Becker, who works for Motorola's RISC Division, led the Motorola designers that were part of the 601 microprocessor development team. Currently, he is continuing work on future PowerPC products. Previously, he worked for Data General and Netstal Machinery, Switzerland.

Becker holds a BSEE and an MSEE from Texas A&M University.

Michael S. Allen worked with IBM to define the 601 bus interface. He is currently designing the bus interface unit for an advanced PowerPC-based microprocessor at Somerset. Prior to his work in Motorola's RISC Division, he developed integrated microprocessor products at National Semiconductor.

Allen holds a BSEE from the University of Texas at Arlington. He is a member of the IEEE.

John S. Muhich is a senior engineer for the IBM Corporation. He is the co-lead architect of the 601 PowerPC microprocessor and the RISC Single Chip (RSC).

David P. Tuttle is an IBM senior engineering manager and the PowerPC 601 microprocessor project manager. Earlier, he was manager of the RS/6000 processor architecture. He also served as the lead logic designer of the RIOS-1 data cache unit and, later, the design manager responsible for the RIOS-1 data cache unit and storage control unit.

Tuttle holds BSEE and MSEE degrees from the University of Louisville and an MBA from the University of Texas at Austin.

Direct questions concerning this article to Michael C. Becker, Motorola, Inc., 9737 Great Hills Trail, Austin, TX 78759; becker%ibmoto.com@aimgate.sps.mot.com.; or to Charles R. Moore, IBM Corp., 9737 Great Hills Trail, Austin, TX 78759; cmoore%ibmoto.com@aimgate.sps.mot.com.

Spearmints:

Hardware Support for Performance Measurements in Distributed Systems

Spearmints permits fine-grained performance measurements in distributed systems with only minimal impact on the observed system. Each machine of the target system must have one sensor that collects relevant events and marks them with global time stamps. The sensors can be attached to a common measurement system that samples the marked events on- or offline, orders them chronologically, and analyzes the resulting sequence.

Uwe Kleinhans

Joerg Kaiser

Karol Czaja

German National Research Center for Computer Science

Distributed systems provide enhanced reliability, increased performance, and improved resource utilization. To take full advantage of these benefits and tailor a system to a given environment, users need detailed characteristics about the system's behavior. However, it is difficult to gain insight into the interaction and performance characteristics, because there is no global control of the activities in these systems.

Several projects used a method called event-triggered measurement to overcome this difficulty. Lange provides a detailed description of this method.[1] Generally, event-triggering provides detailed insight into a certain aspect of a system's behavior by analyzing a chronologically ordered sequence of logically triggered events. The time and place of each event must be known.

Researchers in the Incas project[2] at the University of Kaiserslautern, Germany, observe their system to detect performance problems as well as to support debugging in a distributed environment. To capture process-specific statistics (idle times, blocked times) as well as execution times for procedures and single statements, they analyze process-specific events (start, stop, block), procedure-specific ones (call, return), and executions of single statements.

Another example of an event-triggered measurement system is the Zaehlmonitor 4 monitor developed at the University of Erlangen-Nuernberg, Germany.[3,4] These researchers measure the duration of different phases in communication protocols and parallel algorithms by capturing the calls of and returns from the respective procedures as relevant events. To analyze the synchronization of tightly coupled processes and their communication, they observe single accesses to shared variables or to addresses used as semaphores for critical regions.

The Jewel measurement system focuses on an additional aspect in performance measurement that requires high-resolution observations.[5] Jewel was developed in the Relax project at the Gesellschaft fuer Mathematik und Datenverarbeitung (GMD), the German National Research Center for Computer Science. It was used in the European Space Agency's Dosval project to examine different distributed operating systems for their usability in spacecraft.[6] Jewel provides mean values about a system's performance in a number of events in a specific period of time. Furthermore, Jewel captures individual events to calculate empirical distribution functions, standard deviations, and variations for the analyzed events. Besides the overall performance, these numbers describe the reliability of a system's behavior—an important factor if the observed system is to be used under extreme conditions or in sensitive environments.

Low interference

In the field of system measurement it is crucial that the influence of the measurement on the

0272-1732/93/1000-0069$03.00 © 1993 IEEE

observed system's behavior be as low as possible. The degree of interference in the measured system, the so-called object system, should be very low so that neither the observation nor the running system itself is affected. Accordingly, pure software monitors are not suitable, since they perform the tasks of event-triggered measuring by executing appropriate code within the object system. For that reason, the projects we've described make the tasks of collecting, ordering, and analyzing events part of a measurement system that is separated from the object system. The code to evaluate and process events executes on standard hardware platforms separated from the object system. When an event occurs in the object system, the measurement system receives all the information necessary to identify and evaluate it. As a consequence, the interference in the object system is reduced to the overhead produced by signaling events to the measurement system.

Global time

A prerequisite for the chronological ordering of events in distributed measurement is the notion of global time. To derive performance characteristics, we must be able to calculate the duration between events independently of their places of occurrence. Therefore, each event's moment of occurrence must be fixed according to a global time reference. Obviously, the resolution and accuracy of this time base directly affect the granularity and accuracy that can be achieved for the measurements. Researchers in the mentioned projects show that measurements in distributed systems often imply the observation of intervals with a duration in the range of the execution time for single statements. As standard time measurement facilities do not provide this resolution even for local measurements, they have developed dedicated hardware clocks. For the Jewel system we developed a system of local clocks that are synchronized to provide a global accuracy of about 0.5 μs.

Global measurement support

Technical solutions to the problems of low-interference event detection and global time cannot be developed independently from each other. The way in which events are transferred to the measurement system not only determines the impact on the object system but also seriously affects the granularity as well as the accuracy of time measurements. The subsystem that samples events and the respective time stamps must impose minimal overhead on the object system. It also must guarantee that a time stamp denotes the time of an event's occurrence in the object system as accurately as possible. Any inaccuracy in this subsystem contributes directly to the inaccuracy of the whole measurement.

To provide the items of low interference and high accuracy for various combinations of an object system and a measurement system, we developed a component we call Spearmints, a sensor for performance measurements in distributed systems. Spearmints is a means of gathering descriptions of the events in an object system, while requiring only minimal overhead within the observed system itself. It integrates an efficient mechanism for sampling events and a global time reference that causes minimal interference to the object system while supporting high-resolution measurements. The Spearmints distributed sensor receives events, marks them with global time stamps, and buffers these event records for further processing. It can be easily adapted and interfaced to an object system, and, as it is not tailored to a certain measurement system, the buffered event descriptions can be read by whatever measurement system is used for event processing.

System description

In designing the Spearmints hardware-based sensor, we focused on the essential issue of minimizing the impact of measurement on a system under test. We considered how best to generate complete event records, implement Spearmints as a distributed component, connect it to a local CPU, and interface it to a measurement system.

Generating event records. The contents of the event records generated by Spearmints depend on the information that is collected and stored for sufficient characterization of an event. In most approaches to event-triggered system measurement, an event is characterized by the time and the location of its occurrence as well as by information about its type and actual measurement data.[2,3,5-7] The amount of information that has to be transferred from the object system to the measurement system should be as small as possible to meet the crucial objective of low interference.

The measurement system itself can fetch the time and location of an event's occurrence because they are visible from the outside of the object system. As this approach causes no impact on the system under test, most of the existing measurement systems work this way. The location of any event signaled to the measurement system is the place it is signaled from, while the event's time is gathered from a clock in the interface component that is accessed to signal an event. As the time stamps are used to order and compare events, the clock must produce them according to a global time reference. (The box summarizes GTR basics.)

In general, the type of an event and the actual data of interest associated with it depend on the current state of the object system. The only way to gather this information without any effect on the object system is to use a pure hardware monitor for observing the physical connections in the system under test. This approach is based on the assumption that all relevant events and measurement data can be derived from signal patterns appearing on the physical connections (pattern-matching triggering). Upon recognizing a signal pattern that indicates an event, the hardware monitor stores the pattern and associated data as the description of the event, along

Basics for a global time reference in distributed systems

Generally, system time measurement involves the identification of discrete points of time on a time axis and calculation of the intervals between them. The time axis is unique in the system and starts at one distinguished starting point T_0. Each point X of time on this axis is identified by the time $T(X)$ that has passed since T_0. We measure $T(X)$ by summing the pulses generated from an oscillator with a nominal frequency f_{nom} in the interval from T_0 to X. With $n(X)$ being the count of pulses, $T'(X)$, the measured time at point X is given by $T'(X) := n(X) * f_{nom}^{-1}$.

This leads to a least digit inaccuracy f_{nom}^{-1}, which is ignored in the following, because it is inherent to all kinds of measurements. As every oscillator deviates by factor f from its f_{nom}, $T(X)$ differs from $T'(X)$ by factor f. For every oscillator, a factor F that denotes the oscillator's maximal deviation f from its f_{nom} is given. Therefore, measured time $T'(X)$ is subject to an inaccuracy of

$$T'(X) * (1 - F) \leq T(X) := n(X) f_{nom}^{-1} * (1 + f) \leq T'(X) * (1 + F)$$

To measure the length $t(i)$ of an interval i, we calculate the time between its starting point S_i and its termination point T_i. Assuming an oscillator's deviation f to be stable, the measured values $T'(S_i)$ and $T'(T_i)$ both differ by the same factor f from their real values $T(S_i)$ and $T(T_i)$. Consequently, the measured length $t'(i)$ differs by the factor f from the interval's real length $t(i)$. This results in an inaccuracy $t'(i) * F$ for the measured length:

$$\begin{aligned}
&S_i: && T(S_i) := n(S_i) * f_{nom}^{-1} * (1 + f) = T'(S_i) * (1 + f)\\
&T_i: && T(T_i) := n(T_i) * f_{nom}^{-1} * (1 + f) = T'(T_i) * (1 + f)\\
&t'(i): && t'(i) := T'(T_i) - T'(S_i)\\
&t'(i): && t(i) := T(T_i) - T(S_i) = t'(i) * (1+f) \rightarrow\\
& && \quad t'(i) * (1 - F) \leq t(i) \leq t'(i) * (1 + F)
\end{aligned}$$

A global time reference that is to be used throughout a distributed system to measure time with an accuracy of A, must implement the following two items:

- T_0 is the unique starting point for time measurement throughout the system.
- The duration $t'(i)$ measured for any interval i must not differ by more than A from the duration $t(i)$ elapsed in reality. This must hold independently from the sites where its starting point S_i and its termination point T_i are fixed.

An event's time stamp should denote its time of occurrence as accurately as possible. Using a central clock as a global time reference subjects the time for the transmission of the time stamp to the event's location to significant variations that directly contribute to the measurement's inaccuracy. The variations depend on the communication subsystem used and may even be unpredictable (for example, Ethernet). To avoid these variations, local clocks in the nodes of the distributed system must generate the time stamps.

Due to physical effects that restrict the transmission of high-frequency signals over longer distances, the local clocks cannot use a central oscillator. Therefore the problems of start delay and differing real frequency have to be tackled to implement a global time reference using a system of independently triggered local clocks.

Start delay. Because of the finite speed of communication, two clocks J and K cannot be started simultaneously, but they differ by a specific value $\Delta t_{start}(J,K)$. If T_i and S_i of some interval i are fixed with clocks J,K, the difference $T'(T_i) - T'(S_i)$ has to be corrected by $\Delta t_{start}(J,K)$ to get the length $t'(i)$: $t'(i) = T'(T_i) - T'(S_i) + \Delta t_{start}(J,K)$. With U_{start} being the maximum uncertainty in the knowledge of all values $\Delta t_{start}(J,K)$ throughout the system of local clocks, each measured duration $t'(i)$ is subject to an inaccuracy of $\pm U_{start}$.

Differing real frequency. As every oscillator deviates by a specific factor from its nominal frequency, the times $T'(T_i)$ and $T'(S_i)$ measured with clocks J,K differ by different factors f_J and f_K from the real values $T(T_i)$ and $T(S_i)$. The measurement of $t(i)$ follows:

$$\begin{aligned}
t(i) &:= T(T_i) - T(S_i)\\
&= T'(T_i) * (1 + f_J) - T'(S_i) * (1 + f_K)\\
&= t'(i) + T'(T_i) * f_J - T'(S_i) * f_K
\end{aligned}$$

continued on next page

with a time stamp and an eventual location identifier. This method allows the hardware monitor to transparently recognize an event in the object system.[7]

However, pure hardware monitoring is not suited for general measurements, particularly for those in higher system levels. Because virtual memory management and context switches are transparent to a hardware monitor, it has no information about the context of a particular pattern observed. Thus, relying solely on the observation of physical signal patterns is inadequate for recognizing events that are relevant on higher system levels. Instead, the object system must provide the measurement system with all the information that is necessary to identify these events. For this reason the code of the object system must be equipped with I/O rou-

Basics (cont.)

As there is a given upper bound $\pm F_{max}$ for all oscillators in the system, the oscillator's different deviations from the nominal frequency lead to a maximal inaccuracy $D_{freq}(i) = 2t'(i) * F_{max}$ in measuring an interval i.

To meet the requirements for a global time reference with an accuracy of A, the system of local clocks must contain mechanisms that guarantee that the following condition is not violated for any pair of clocks and any interval i:

$$| t(i) - t'(i) | \leq 2t(i) * F_{max} + U_{start} < A$$

The common start of all local clocks must guarantee the upper-bound U_{start} for the uncertainty, to which the delay in the common start of some pair (J,K) of clocks is known. U_{start} has to be small in relation to the required accuracy of A. Intending a resolution of less than 1 µs, as used in today's measurement systems, the common start must be implemented using a communication system with almost definite transmission delays (point-to-point connections, Token Ring).

Synchronization is necessary to enforce that for any interval i the deviation $D_{freq}(i)$ of any pair of clocks is limited to $D_{freq}(i) = 2 * t'(i) * F_{max} < A - U_{start}$. We achieve this by synchronizing the clocks after the common start with a fixed frequency f_{syn}. After adjusting to the time $n * f_{syn}^{-1}$ on the last synchronization, we set every clock to the time $(n + 1) * f_{syn}^{-1}$ when the next synchronization occurs. Thus, each pair of clocks can deviate only in the interval between two consecutive synchronizations. This limits the clocks' maximum deviation to $D < 2 * F_{max} * f_{syn}^{-1}$. Assuming a maximal frequency inaccuracy of F_{max}, we synchronize the clocks with a frequency f_{syn} of $f_{syn} > 2 * F_{max}/(A - U_{start})$ to achieve a resolution of A.

tines at those points where relevant events are to be signaled (embedded-code triggering). Upon executing such a routine, the object system writes both an event's type and its associated measurement data to the measurement system. Spearmints receives this information and combines it with the event's time. The resulting event record comprises all data that is necessary to analyze the event and stores it for further processing by the measurement system.

Spearmints as a distributed component. According to the discussion in the Basics box, the global time reference must be implemented as a distributed component to minimize any inaccuracy of the events' time stamps. Furthermore, the overhead in the object system, which is necessary to transfer information about an event to Spearmints, determines the degree of interference due to signaling an event. Providing only insufficient bandwidth or using an interface with long access times may result in a significant reduction of the object systems' performance. Consequently, Spearmints consists of multiple sensors, each of them interfaced to one node of the object system. Every sensor contains a local clock that is synchronized with the clocks of the other sensors to constitute a global time reference. If an event occurs on a node, the object system transfers the appropriate infomation to the local Spearmints sensor that generates the event record and stores it. The sensors connect to the measurement system that fetches the event records for further processing. As an event record comprises all information about an event, this transfer is no longer critical to the measurement's accuracy.

A Spearmints sensor and the object system. Current distributed systems generally place nodes on workstations. Their architectures provide two classes of interfaces that meet the conditions necessary for transferring information from a node of the object system to the local component of Spearmints. They are the CPU and the system bus interfaces.

Connecting Spearmints directly to the local CPU in the object system's nodes yields a minimal delay for signaling an event. If the CPU interface is not arbitrated for multiple masters, this delay only results from restrictions for transmission over the physical connections. As it is small in relation to the magnitude of the accuracy desired for the measurements, the transfer over the CPU interface does not impact the time stamp's accuracy.

To use simple store instructions in the object system to signal events, the local Spearments sensor must be integrated into a local address space that is visible and can be written to in all modes of computation. Using today's RISC CPUs, events can be signaled using store instructions with an overhead of 1 cycle per byte. Experiments with an object system signaling events with a rate of about $5 * 10^{-4}$ events/cycle and an data rate of 8 bytes/event[5,6] show that the measurements lead to a negligible overhead of about 0.5 percent for event signaling. As a result, connecting a component of Speamints to the local CPU's interface contributes to a very high degree to the objectives of low interference and high resolution.

In most of the hardware platforms used today, it is quite easy to attach the sensor to the system bus such as an Sbus, Turbochannel, or VMEbus. As long as this connection is exclusively used by the CPU and the sensor can be mapped into a directly accessible address space, the system bus may be used as well as the CPU interface. Using an address space that can be accessed only by system calls or explicit I/O routines results in an additional overhead for calling and executing this additional code to signal events. This overhead adds to the impact on the object system resulting from the measurement. Furthermore the variation in the execution times of system calls or I/O routines adds to the inaccuracy of the events' time stamps.

If the system bus is arbitrated between different components of the system, there are periods when an event cannot be signaled instantaneously because the bus is reserved for exclusive use by another component. If an event occurs during such an interval, the delay in signaling the event depends on the strategy employed in the channel's arbitration. As the arbitration strategies are not preemptive in general, the time to signal an event to the local Spearmints sensor may be subject to large variations that contribute directly to the inaccuracy of the event's time stamp. As the bus is not arbitrated, during, for example, burst transfers, an event signaled in this interval may be delayed by up to the duration of one burst transfer. This limits its time stamp's accuracy to the length of a burst. Typical values for a burst transfer from the disk to memory are 800 µs in a VMEbus workstation and 5 µs in a more sophisticated DEC 5000/200 system. This maximal delay in signaling an event not only influences the time stamp's accuracy but also slows down the object system's activities, thus implying a higher degree of interference.

Consequently, to achieve maximum accuracy for the time stamps and minimal interference in the system under test, the Spearmints sensors should be connected as closely as possible to the CPUs in the object systems' nodes. The described effects of bus arbitration have to be considered if the system bus of a node is used to integrate a Spearmints sensor into an object system. Generally, every Spearmints sensor should be mapped into an address space that can be directly accessed by every code running on the local node of the object system.

Spearmints sensor and the measurement system. The transfer of event records from the Spearmints sensors to the measurement system is not critical to the accuracy of the measurements and the impact on an object system. Instead, the frequency of events occurring in the object system guides the design of the interface between a Spearmints sensor and a measurement system. Its bandwidth has to be large enough to transfer the event records from the sensor to the measurement system before the sensor is flooded with event records. Furthermore, the measurement system uses this interface to program the sensors. Generally, the sensors should have an interface with sufficient bandwidth for bidirectional data exchange with a large variety of measurement systems running on various hardware platforms.

Architecture

According to these concepts, we implemented Spearmints as a system of sensors that constitute an interface between an object system and a measurement system. Each Spearmints sensor is attached to a node of the object system and to the measurement system. Furthermore, the local clocks of the sensors are linked by a synchronization channel that is necessary for the reliable and correctly timed transmission of synchronization signals. This is a prerequisite for the synchronization scheme used to implement Spearmints' global time reference.

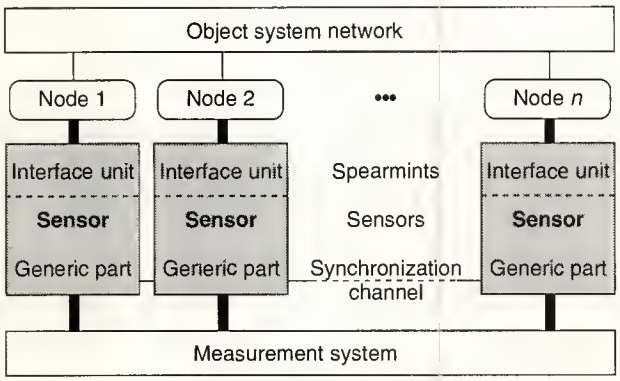


Figure 1. Spearmints' placement between an object system and a measurement system.

The overall architecture of Spearmints is shown in Figure 1.

A Spearmints sensor consists of an interface unit and a generic part. This separation provides the flexibility to integrate the sensor into different kinds of nodes. The interface unit depends on the architecture of the object system's node to which the sensor is attached. The generic part of a sensor executes the functions of creating and storing event records as well as passing them to a measurement system.

The different interface units mainly adopt standard bus interfaces and protocols. A sensor connected to these buses will be easily and quickly accessible from an object system, because they cover address spaces that can be mapped into the address spaces of user processes. If the object system writes to the interface unit of this sensor, the unit supplies the generic part with the address, the data, and a control signal that enables the sensor.

The generic part. Figure 2 (next page) depicts the structure of the sensor's generic part. This part is enabled by the sensor's interface unit when the object system signals an event by writing to an address that is assigned to the sensor.

As in many measurement systems,[2,3,5,7] a part of the address that is accessed by the object system is interpreted as the signaled event's class. Thus, different event types can be distinguished. To support this feature, every Spearmints component occupies a range of consecutive addresses on the bus. Each access to this address space is interpreted as the signaling of an event. The address offset within this address space is interpreted as the event's type identifier.

Often an object system is observed under various aspects by doing multiple measurements, each of them focusing on different types of events. As it would be too cumbersome to change the instrumentation of an object system's code whenever a new aspect is to be observed, most systems that are subject to measurements are monitored continuously. That is, their code contains I/O routines for all kinds of events that

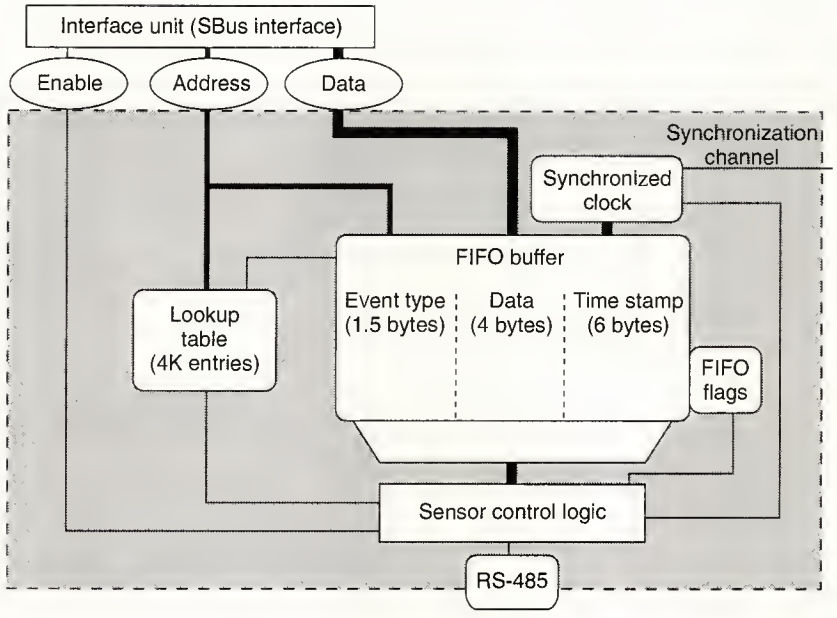


Figure 2. Structure of a sensor's generic part.

Table 1. The effect of our synchronization scheme.

Duration D (seconds)	Clocks not synchronized		Clocks synchronized	
	Deviation Δt	$F = \Delta t/D$	Deviation Δt	$F = \Delta t/D$
30	$2{,}803 \times 10^{-7}$	9.3×10^{-6}	0	0
300	$28{,}915 \times 10^{-7}$	9.6×10^{-6}	0	0
3,600	0.0344	9.5×10^{-6}	0	0
61,200 (17 hr.)	0.6	9.8×10^{-6}	10^{-7}	1.6×10^{-12}

may be of relevance for any measurement. The measurement system marks irrelevant event types in each Spearmints sensor's lookup table to prevent itself from being swamped with needless information. The sensor differentiates up to 4,096 types of events.

Upon receiving an event's type identifier, the generic part uses the lookup table to verify that it is a relevant event. If an event is relevant, the event's type identifier, the transmitted data word, and the actual count of the local clock are combined into one event record. The record is stored as one entry in the sensor's dual-ported FIFO buffer.

As Spearmints is intended to be used in today's workstations, we assumed a word length of 32 bits for the received data. If an event requires more than 4 bytes of data, additional accesses to the sensor become necessary. Dedication of a special event type for these accesses allows the resulting entries in the FIFO buffer to be recognized as additional data for a previously signaled event. The FIFO buffer provides multiple flags that indicate the degree to which the buffer is filled. These flags can be polled by the measurement system that empties the buffer before an overflow can occur.

The local clocks of all Spearmints sensors constitute a global time reference with a resolution of 100 ns (see adjacent box). The clock's count width of 48 bits enables the generation of unique global time stamps for more than 7,800 hours. A bidirectional serial interface (RS 485) provides the communication channel between the sensor and the measurement system. With an assumed mean frequency of events of about $10^4/s$[2,5,6,8] and an average of 11.5 bytes of data associated with one event,[5] the transfer of event records takes an approximate channel capacity of about 115 Kbytes/s. As bursts of events are buffered in the FIFO buffer, the serial interface's capacity of about 1 Mbyte/s meets this requirement. The sensor control logic controls the communication between the sensor and the measurement system via the serial interface. It interprets received messages and generates the appropriate control signals for the sensor's components.

Evaluation of the GTR. To evaluate and test Spearmints' global time reference, we have implemented a prototype consisting of a master-slave pair that is integrated into two DEC 3100 workstations. The clocks contain a 10-MHz oscillator, an 82C54 programmable counter, and two PAL22V10s containing the logic for synchronization and control. A simple fiber optics system serves as the synchronization channel.

To suffer as little access latency as possible, we attached the clocks to the workstations' EPROM sockets, which are directly connected to the CPU bus. We used the two clocks for measuring various intervals to verify our synchronization algorithm. We measured the duration D of each interval with the clocks running independently as well as being synchronized. For both cases we calculated the absolute deviation Δt of the clocks and the relative deviation $F = \Delta t/D$. The results are shown in Table 1.

Spearmints' global time reference

Each of Spearmints' sensors comes with a local clock that measures time by counting the pulses generated by the clock's oscillator running at frequency f_{nom} of 10 MHz. (See Figure A). To constitute a global time reference, the clocks exchange appropriate signals via a dedicated synchronization channel. The literature presents various methods for synchronizing clocks. They can be generally classified as either software synchronization based on the exchange and evaluation of messages or hardware synchronization using physical signals and effects.[2-4]

Software synchronization generally provides more flexibility regarding the clocks' interconnections. However, the software-based approaches presented up to now do not achieve a global resolution better than 100 µs.[1] This is mainly due to the communication necessary for software synchronization being quite complex and time consuming. Furthermore, many communication schemes cannot even guarantee an upper bound U_{start} for the uncertainty in the delay of the common start and the synchronization (Ethernet).

To implement Spearmints' global time reference, we modified a hardware-based method mentioned in Zieher and Zitterbart,[3] which is based on the idea of having one master clock and several slave clocks in the system. After the common start of all clocks, the master clock sends a synchronization signal with a frequency of f_{syn}, causing the slave clocks to synchronize themselves to the master.

As we wanted to trade the synchronization channel's reliability for its costs, we extended this synchronization scheme to tolerate the loss of a certain amount of synchronization signals. The measurement system executes the Reset instruction to prepare the master clock for global time measurement. The master clock disables its counter, resets it, and generates the RESET signal. Upon receiving this signal, the slave clocks lock and initialize their counters. Upon receiving a Start instruction, the master clock unlocks its counter and makes the slaves do the same by emitting the START signal. Once all clocks are started, the master clock generates the two synchronization signals SYNC-A and SYNC-B.

Additionally, we equipped the system of local clocks with a global stop mechanism. Executing a global stop instruction for the master clock lets this mechanism suspend further counting and generate a STOP signal that halts the slave clocks. Between a global start and a global stop, the clocks are synchronized according to Spearmints' synchronization algorithm. This algorithm guarantees that no pair of clocks deviates by more than $2 * F_{max} * f_{syn}^{-1}$, if $F_{max} < 0.5$ (usually, relevant oscillators have an $F \leq 10^{-5}$):

- The master clock generates the synchronization signals alternatingly with frequency f_{syn}. The nominal duration $T_{syn} = f_{syn}^{-1}$ between two successive synchronizations is known throughout the system of clocks.
- At time $2n * T_{syn}$, with $n = 0,1,2, \ldots$, the master clock generates signal SYNC-A. At time $(2n + 1) * T_{syn}$, with $n = 0,1,2, \ldots$, it generates signal SYNC-B.
- Every time a slave clock receives a synchronization signal, its reaction depends on the received signal and its actual local time t_s (see Table A).
- As long as a slave clock does not receive a synchronization signal, it runs locally.

This algorithm guarantees that the clocks in the system resynchronize to a unique global time, if no more than $1/2 * F_{max}^{-1} - 1$ consecutive synchronization signals have been lost. With t (mas) being the time at the master clock when synchronization comes up again, successful resynchronization takes place as follows:

continued on next page

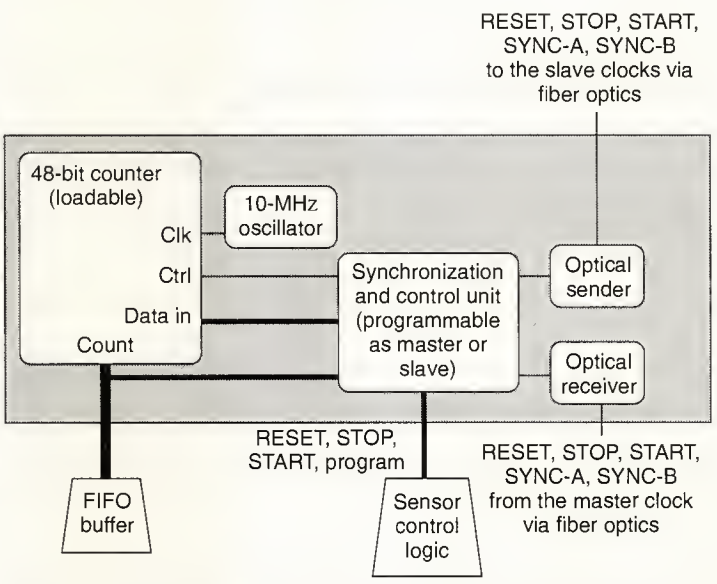


Figure A. Local clock of Spearmints' global time reference.

Spearmints' global time reference (cont.)

Table A. Signal reactions.

If signal is	and if $n \geq 0$ exists such that	then slave is	and t_s has to be adjusted to
SYNC-A	$0 < (2n-1) * T_{syn} \leq t_s < 2n * T_{syn}$	Too slow	$t_s := 2n * T_{syn}$
SYNC-A	$2n * T_{syn} \leq t_s < (2n+1) * T_{syn}$	Ok or too fast	$t_s := 2n * T_{syn}$
SYNC-B	$0 < (2n-1) * T_{syn} \leq t_s < 2n * T_{syn}$	Ok or too fast	$t_s := (2n-1) * T_{syn}$
SYNC-B	$2n * T_{syn} \leq t_s < (2n+1) * T_{syn}$	Too slow	$t_s := (2n+1) * T_{syn}$

- With $t(\text{mas}) = 2 * l * T_{syn}$ ($l = 0,1,2, \ldots$), the master generates signal SYNC-A. Every slave clock adjusts itself to the time $2 * l * T_{syn}$, if the following condition holds for its local time $t(\text{slv})$: $(2 * l - 1) * T_{syn} < t(\text{slv}) < (2 * l + 1) * T_{syn}$.
- With $t(\text{mas}) = (2 * l + 1) * T_{syn}$ ($l = 0,1,2, \ldots$), the master generates signal SYNC-B. Every slave clock adjusts itself to the time $(2 * l + 1) * T_{syn}$, if the following condition holds for its local time $t(\text{slv})$: $2 * l * T_{syn} < t(\text{slv}) < (2 * l + 2) * T_{syn}$.

In both cases the condition for correct resynchronization requires $|t(\text{mas}) - t(\text{slv})| < T_{syn}$. As each pair of clocks deviates by a maximum rate of $2 * F_{max} * t$, this condition holds if the synchronization is not down for longer than $F_{max}^{-1} * T_{syn}/2$. That is, up to $1/2 * F_{max}^{-1} - 1$ synchronization signals may be lost without affecting a successful synchronization.

Its synchronization and control logic controls each local clock. By setting a flag in the control logic, the clock can be programmed as the master clock or as a slave clock of the global time reference. Depending on this flag, the control logic generates control and synchronization signals, or it receives them and adjusts the clock's counter accordingly. The master clock sends the serially coded signals via fiber optics to the slave clocks. Including the variations in transmission times and the different switch times of the clocks' logic, the uncertainty in the signals' delay does not exceed 30 ns. Assuming an inaccuracy of the oscillators' frequency of $F_{max} * 10^{-6}$, the clocks are synchronized with a frequency of $f_{syn} = 29$ Hz to achieve the 100-ns accuracy of the global time reference.

References

1. Flaviu Cristian, "A Probabilistic Approach to Distributed Clock Synchronization," *Proc. Ninth Conf. Distributed Computing Systems*, 1989, p. 288.
2. R. Hofman et al., "Integrating Monitoring and Modeling to a Performance Evaluation Methodology," *Entwurf und Betrieb verteilter Systeme*, Springer-Verlag, Berlin, 1990, pp. 122-149.
3. M. Zieher and M. Zitterbart, "NETMON—A Distributed Monitoring System," *Proc. EFOC/LAN, Sixth European Fibre Optic Communications and Local Area Networks Exposition*, IGI Europe, June/July 1988, pp. 452-457.
4. Hermann Kopetz, "Clock Synchronization in Distributed Real-Time Systems," *IEEE Trans. Computers*, Vol. C-36, No. 8, Aug. 1987, pp. 933–940.

The comparison of the results underlines the necessity for clock synchronization. Running the clocks with the synchronization lets us experience a constant difference of 200 ns or 300 ns in the values gathered from the two clocks. This results from a constant delay of 200 ns in the start of the clocks and the least significant digit error that is inherent to all kinds of digital measurement. The absolute deviation between two unsynchronized clocks is a linear function of the measured time. The relative deviation equals the nominal inaccuracy F_{max} of the used oscillators. This experiment shows that our synchronization algorithm is suitable to achieve a global accuracy in the range of a few hundred nanoseconds.

Moreover, we tested the robustness of Spearmints' synchronization scheme against the loss of sychronization signals. After starting the two clocks globally, we suppressed the generation of synchronization signals for some intervals. We experienced a steadily increasing deviation of the clocks. If the synchronization came up again within five minutes, the two clocks are resynchronized on a common global time and run synchronized again. This value matches the theoretical limit $1/2 * F_{max}^{-1} * T_{sync}$ for the tolerable duration of failing synchronization very well. In our experiments, the deviation

in frequency and the time between two succeeding synchronizations had values of $F_{max} = 10^{-5}$ and $T_{sync} = 65{,}536 * 10^{-7}$s. Consequently, we could tolerate the loss of synchronization signals for a maximum duration of about 5.5 minutes.

SPEARMINTS IS A SYSTEM OF HARDWARE COMPONENTS that can be easily interfaced to the nodes of an instrumented distributed system for monitoring or evaluation using event-triggered measurements. The design of Spearmints follows the idea of providing a simple and universal tool that causes little interference and furnishes highly accurate measurements in distributed systems. As Spearmints only requires standard interfaces for its integration into an object system and its connection to a measurement system, it permits a wide range of measurement systems for the evaluation of a variety of distributed systems. This versatility distinguishes Spearmints from other similar approaches that mostly are tailored to a certain object or a certain measurement system.[2,4,7,9]

Our actual work focuses on the implementation of the sensor and its prototypical integration in GMD's Jewel measurement system. As far as Spearmints' global time reference is concerned, we used experiences from a prototype integrated into two DEC 3100 workstations and one that is already used by the Relax measurement system[10] to perform measurements in VMEbus-based systems. These measurements take advantage of 640-ns-resolution global time stamps. Actually, GMD develops an ASIC version of the global time reference for the Spearmints' prototype, which aims at an overall resolution of about 500 ns. To achieve this using a nominal clock frequency of 10 MHz and a synchronization frequency of $F_{syn} > 30$ Hz, the sensors must be integrated into the object system via an interface that guarantees a maximum uncertainty in access times of about 250 ns. GMD plans to integrate this new generation of Spearmints sensors into Turbochannel and Sbus workstations.

Moreover, we are interested in the question of how the redundancy in the system of synchronized local clocks and the ability to detect synchronization faults as well as to recover from them can be exploited to provide a fault-tolerant global time base. Making the sensor's clocks accessible to the object system may result in a fault-tolerant global time base of high accuracy that can be used in the distributed object system as well as in the measurement system. Furthermore, a fault-tolerant global time base of high resolution may be of great advantage for distributed real-time systems. How Spearmints' global time reference can be used in this context is part of the present work.

References

1. Frank Lange, "Leistungsmessung in verteilten Systemen—Konzepte und Werkzeuge zur Instrumentierung," (Performance Measurement in Distributed Systems—Concepts and Tools for Instrumentation), GMD Studien No. 179, May 1990.
2. Dieter Wybranietz and Dieter Haban, "Monitoring and Performance Measuring Distributed Systems Under Operation," *Proc. ACM SIGMETRICS Conf. Measurement and Modelling of Comp. Systems,* Association of Computing Machinery, New York, 1988, pp. 197–206.
3. R. Hofman et al., "Integrating Monitoring and Modeling to a Performance Evaluation Methodology," *Entwurf und Betrieb verteilter Systeme* (Development and Operation of Distributed Systems), Springer-Verlag, Berlin, 1990, pp. 122-149.
4. R. Hofman et al., "An Approach to Monitoring and Modeling of Multiprocessor and Multicomputer Systems," *Proc. Int'l Seminar on Performance of Distributed and Parallel Systems*, 1988, pp. 91–110.
5. F. Lange, R. Krueger, and M. Gergeleit, "JEWEL: Design and Implementation of a Distributed Measurement System" *IEEE Trans. Parallel and Distributed Systems,* July 1992.
6. F. Lange et al., "Distributed Systems Validation Method —Final Report," GMD, Studien No. 208, 1992.
7. A. Mink et al., "Multiprocessor Performance Measurement Instrumentation," *Computer*, Sept. 1990, pp. 63-75.
8. Cui-Qing Yang and Barton P. Miller, "Performance Measurement for Parallel and Distributed Programs: A Structured and Automatic Approach," *IEEE Trans. Software Engineering*, Vol. 15, No.12, 1989, p. 288.
9. Flaviu Cristian, "A Probabilistic Approach to Distributed Clock Synchronization," *Proc. Ninth Conf. Distributed Computing Systems*, 1989.
10. Reinhold Krueger et al., "The RelaX Concepts and Tools for Distributed Systems Evaluation," GMD Studien No. 168, Oct. 1989.

Uwe Kleinhans works in the field of antivirus systems at Norman Data Defense Systems AS in Drammen, Norway. This article reflects his work at the German National Research Center for Computer Science, Gesellschaft fuer Mathematik und Datenverarbeitung (GMD), in St. Augustin, Germany. His main research activities lay in the field of

system architecture, where he analyzed the software-hardware interface for problems of fault tolerance, security, and system measurement in distributed and object-oriented systems. To show how conventional hardware can be enhanced with regard to dedicated requirements resulting from this analysis, he developed different hardware components and implemented them prototypically using ASIC technology.

Kleinhans studied computer science at the University of Bonn, where he received his diploma.

Joerg Kaiser heads the CREW (Cooperative Engineering Workbench) project at GMD in St. Augustin; CREW concerns system support for cooperative group work in design and engineering applications. His research work involves computer architecture and operating systems with an emphasis on object orientation and fault tolerance. He has published over 20 papers in these areas, investigated fault-tolerant VLSI structures, and held responsibility for the design and implementation of hardware support for an object-oriented distributed computer system. He has also worked in the area of high-precision and low-interference system monitoring and measurement.

Kaiser receved his diploma in computer science from Bonn University.

Karol Czaja works as an MS systems adviser at Pfeifer and Langen, Cologne, Germany. At the time this article was written, he was a researcher at GMD where he developed an access control monitor providing protection in persistent object-oriented systems. His main research interests include computer architecture and operating systems with an emphasis on object orientation and fault tolerance.

Czaja received his diploma in computer science from the University of Bonn.

Direct questions concerning this article to Uwe Kleinhans, Kristian Bogneruds vei 8, N-0956 Oslo, Norway; or mettek@novell.oih.no.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 165 Medium 166 High 167

How Does Processor MHz Relate to End-User Performance?

Part 2: Memory Subsystem and Instruction Set

We conclude our study of end-user performance begun last issue by describing the performance implications of the memory subsystem and the effect of instruction sets on path length. Measured performances on many systems support our initial claim that cycle time is not sufficient to determine performance.

Steven W. White

Phil D. Hester

Jack W. Kemp

IBM, Advanced Workstations and Systems Division

G. Jeanette McWilliams

Performance InDeed!

In Part 1 we studied end-user performance by selecting two microprocessors that used distinctly different design philosophies to achieve comparable performance levels. The 62.5-MHz IBM RISC System/6000 Model 580[1-3] (called RS1 here) exemplified the instruction-level parallelism approach, while the 160-MHz DEC Alpha[4-8] illustrated the aggressive clock rate approach. We defined the performance components as clock rate, cycles per instruction (CPI), and path length. After discussing clock rate, we examined the differences within and across functional units and how they affected CPI. Now, we describe the performance implications of the memory subsystems and the instruction sets' effect on path length.

CPI, memory subsystem effects

Part 1 assumed that all instructions and data reside in the first-level (L1) caches. It also assumed that the virtual address translation information required to access instructions and data resides in the translation look-aside buffers (TLBs). Pipeline effects and interactions were lumped into the infinite cache element of CPI. In addition to pipeline effects, cache and TLB miss penalties can contribute significantly to measurable or finite cache CPI.

We define our end-user performance CPI component as

Finite cache CPI= infinite cache CPI
+ finite cache effect + finite TLB effect

The finite cache effect is the product of the average number of cache misses per instruction and the average cache miss penalty in cycles. The finite TLB effect is the product of the average number of TLB misses per instruction and the average TLB miss penalty in cycles. We further partition the finite cache effect between penalties for cache misses and the eventual updating of main memory for stores. The cache and TLB misses include both instruction and data references.

The unit for each of these quantities is cycles per instruction. If each term is multiplied by instruction count, our equation simply states that total measurable execution time is the sum of the pipeline cycles, the cache miss penalty cycles, and the TLB miss penalty cycles. Improvements obtained by reducing either the pipeline CPI or finite cache/TLB effects are amortized across the total execution time.

L1 cache miss counts. In attempting to minimize the number of cache misses, a cache designer may make four main choices: set associativity, line size, capacity, and store policy.

0272-1732/93/1000-0079$03.00 © 1993 IEEE

Table 1. Simulated miss counts as a function of instruction and data cache capacity and geometry.

	Instruction cache		Data cache	
SPECint89 benchmark	Alpha (8 Kbytes, direct, 32-byte line)	RS1 (32 Kbytes, two-way, 64-byte line)	Alpha (8 Kbytes, direct, 32-byte line)	RS1 (64 Kbytes, four-way, 128-byte line)
Gcc	1,361,171	204,268	646,776	32,557
Li	227,991	819	515,430	1,557
Eqntott	86	51	344,742	146,030
Espresso	56,831	1,090	446,409	3,825
Total cache misses	1,646,079 @ 6 cycles	206,228 @ 10 cycles	1,953,357 @ 6 cycles	183,969 @ 10 cycles
Finite cache miss effect/instruction	0.082	0.017	0.098	0.015

High clock rates are more easily achieved by simplifying some aspects of the cache design. Options include smaller caches, direct-mapped caches, and store-through caches (and prohibiting byte and 2-byte accesses as well as unaligned accesses).

To see how clock rate goals affect these choices, consider set associativity. In an *N*-way set-associative cache, designers must compare appropriate address tags to determine which entry contains the requested data prior to its use. In direct-mapped caches, a common strategy is to use the only piece of data in the selected congruence class and start it down the pipeline. Then the processor cancels the operation if it is later determined that a cache miss occurred and incorrect data was being used.

In high cache hit environments the cancellation penalty is rarely paid, and this practice allows the designer a head start on getting the data to the functional unit, effectively shortening the pipeline. The drawback is that direct-mapped caches have higher miss rates than the equivalent set-associative caches, with the difference in miss counts being more pronounced for small (4-Kbyte, 8-Kbyte) caches.[9] Therefore, designers often choose direct-mapped caches when the cache is large (and associativity is not as important) or when access time is critical.

The second key design lever for reducing cache misses is cache capacity. Capacity is often limited by cycle time goals since cache and TLB accesses are often on several critical timing paths. As cycle time is decreased, designers must either add cycles per cache access or reduce the time (in nanoseconds) to access the cache. The former results in more load-use delay as illustrated by the two-cycle delay for the 62.5-MHz RS1 when compared to the three-cycle delay for the 160-MHz Alpha.

Physical and electrical constraints, which worsen as cache capacity increases, limit the access time. We can reduce the physical access time for the cache by decreasing the capacity of the cache. Physical access time can also be reduced by using only on-chip cache to avoid chip crossings; however, this also places a limit on capacity. (The assumed inverse relationship between clock rate and L1 cache capacity is supported by comparing the 32-Kbyte instruction cache (I-cache) and 64-Kbyte data cache (D-cache) on the 62.5-MHz RS1 to the 8-Kbyte instruction and data caches on the 160-MHz Alpha.)

For a given main memory (or L2) with a fixed access time in nanoseconds, increasing the processor clock rate has a compounding effect on cache. It forces designers to smaller caches that have higher miss rates, and it increases the number of *processor* clocks required to satisfy a miss.

Table 1 illustrates the effects of set associativity, line size, and capacity for the four SPECint89[10] codes. (These effects are based on RS1 access patterns and vary with compiler.) The table lists the results of cache simulations for snapshots of the SPECint89 work loads. Our simulations used three trace snapshots (10 million instructions each) from each of the four benchmarks. Traces of this length are adequate to render the cold cache start-up effects negligible; for an 8-Kbyte cache with 32-byte lines, there can be at most 256 additional misses due to cold cache effects. The additional miss penalty is three or four orders of magnitude smaller than the execution time of the trace. (The store instructions were filtered from the traces prior to modeling the Alpha D-cache to account for Alpha implementing an allocate-on-read cache.)

The simulated miss counts reflect RS1 code. Some aspects of the Alpha design have potential for further reducing the effectiveness of the 8-Kbyte I-cache. To fill the load, branch, and functional unit delays, the compiler writer must perform aggressive loop unrolling and in-lining.[6] This increases the footprint or instruction working set (not the dynamic path

length) of the application and can cause more I-cache misses than would occur with tightly rolled loops and subroutine calls. In processors with small caches, the choices may be unattractive: to stall for the pipeline delays or to wait for an I-cache miss to be satisfied. Based on our experiences in moving from the 8-Kbyte I-cache on the original RISC System/6000s (1990) to the 32-Kbyte I-cache on the current RS1, opportunities to gain performance by code expansion are much more limited in the smaller I-caches.

L1 miss penalties. The average L1 cache miss penalty is often the part of the finite cache effect that is most difficult to determine. Due to interaction between the pipelines and caches, not all misses incur equal penalty. If instruction prefetch triggers the miss early enough, and if instructions already fetched can provide work for the functional units while the miss is being satisfied, an I-cache miss is completely hidden. Hidden misses do not result in a penalty.

At the other extreme, a data dependency may cause the pipelines to stall the cycle after a cache miss, resulting in the maximum penalty. Additional functional units may decrease the infinite cache CPI and allow the dispatched work to be completed earlier. However when a cache miss is outstanding, finishing the work earlier may simply mean stalling the pipelines for more cycles. Pipeline and cache effects are not independent; cache and TLB misses can partially or completely negate the benefit of additional units.

Penalties for the cache misses shown in Table 1 can be subdivided into leading-edge and trailing-edge categories. The leading-edge portion includes cycles that elapse between the start of the miss and the first transfer. In both designs, this first transfer contains the requested data (also known as critical word first). The leading-edge time is dependent on main memory (or L2) access time. The trailing-edge includes the remaining cycles of the miss, to transfer the rest of the cache line. Trailing edge is a function of the cache line length, the width of the cache-memory bus, and the number of processor cycles per transfer.

Both designs include a number of features to reduce the penalty of I-cache and D-cache misses. A store-back buffer in RS1 reduces the leading-edge penalty for D-cache misses. When a D-cache miss causes a dirty (or modified) line in the cache to be selected for replacement, main memory must be updated with the modified data. Without a buffer, this memory update, or cast-out operation, must be made prior to the cache miss request so that room in the cache is available for the returning miss data. In RS1, the modified cache line data is placed in the store-back buffer, allowing the cache miss request to be started immediately. The bus and memory transaction for the dirty line is delayed until the bus is idle or the next line is cast out.

RS1 reduces the leading-edge effects of an I-cache miss with branch look-ahead logic (discussed in Part 1). Additionally, the fixed-/floating-point unit queues allow the branch unit to proceed ahead of the arithmetic units. As a result, some of an I-cache miss leading-edge penalty can be overlapped with previously dispatched work.

Alpha saves some D-cache leading-edge cycles with a limited form of hit-under-miss logic. Hit under miss allows subsequent storage reference instructions to access the cache before the first transfer of the missing line arrives. An Alpha D-cache miss condition is recognized when the load reaches stage 6 in the load and store pipeline. (Stage 3 is dispatch.) When a D-cache miss occurs, Alpha blocks issue for a class of instructions that includes loads and stores. The only can-

Performance metrics

When deciding how to judge the performance of a processor design, remember to select work loads that are similar to those of the intended user of the design. We selected the SPEC suite of benchmarks because they include a variety of complete applications from the workstation/server user community. SPEC closely controls source code and mandates full disclosure of variances and configurations. This approach provides a fair basis for system comparisons.

In our study we used metrics based on two SPEC suites. SPECmark89 is the geometric mean of performance on 10 benchmarks, four fixed point and six floating point. In 1992, SPEC released a second suite that carries separate fixed- and floating-point metrics: SPECint92 and SPECfp92. These metrics are geometric means of performance on six integer benchmarks and 14 floating-point benchmarks. With the exception of a matrix multiply program, all members of the original suite are included in the newer one. The original benchmarks were improved for the second suite (portability changes, new input files to provide runtimes long enough to be easily repeatable on newer faster machines, and so on).

Vendors of most systems in the server/workstation market have reported performance on both suites, so we have a good body of reliable data on many systems.

If we were just now starting the work that led to this article, we would rely entirely on SPECint92, SPECfp92, and Linpack to characterize performance. (We would include Linpack because it represents a large body of engineering/scientific applications that the matrix code covered in the old suite.) When we made simulations to support some of our observations, we only had traces on the older suite.

When we present actual performance results for the two architectures under consideration, we show all four metrics (SPECint92, SPECfp92, SPECmark89, and Linpack).

Due to interaction between the pipelines and caches, not all cache misses incur equal penalty.

didates for hit-under-miss processing are the two storage references—if present—in stages 4 and 5 (already issued), if they are D-cache hits. If either is a miss, it is queued in a "silo." All subsequent storage reference instructions are blocked from dispatching until the silo empties and the first transfer from the final miss arrives. This strategy eliminates up to two cycles of the leading-edge penalty.

To hide some leading-edge effect on I-cache misses, Alpha includes an I-cache stream buffer used only for the sequential execution path.[4] When an I-cache miss occurs and the line is not in the stream buffer, Alpha generates a memory (or L2) request. If the line is in the stream buffer, the miss is satisfied sooner. In either case, Alpha generates a prefetch request for the next sequential line. Although this may reduce the leading-edge effects in sequential code blocks, it may increase memory (or L2) use and bus contention with extraneous requests in branch-rich codes such as gcc. (It is not clear to us whether stream buffer requests to the L2 can be canceled if it is determined that a branch has been taken.)

Alpha's trailing-edge penalty may be very small. In one mode, 16-byte transfers per bus cycle (at least two processor cycles) can fill a 32-byte line in four (or more) processor cycles. (Alpha supports bus frequencies between one eighth to one half of the on-chip processor clock.) As 16-byte transfers arrive from the bus, they are moved, 8 bytes per processor cycle, into a "pending fill" latch. During this time, the cache remains accessible to other requests. When the entire line has been accumulated in the pending fill latch, it is dumped into the cache. The cited references do not state whether data (other than the requested operand) is accessible from the pending fill latch before it is transferred to the cache. It is also not clear whether the I-cache or a returning I-cache line is accessible during an I-cache line fill.

RS1 benefits (in the area of cache hits) from long D-cache lines at the expense of a moderate reloading period of eight cycles. To reduce the D-cache trailing-edge penalty, RS1 supports hit under fill with a cache reload buffer (CRB), which is similar to the Alpha pending fill latch. Hit under fill allows subsequent accesses to the cache once the first word returns on a cache miss. Data is accessible from the CRB as it returns. RS1 hides some of the I-cache trailing-edge effect by forwarding the returning instructions to the fetch pipeline as the I-cache fills.

For the SPECint89 traces, simulations provide the I-cache and D-cache miss counts previously shown in Table 1. The performance impact of these misses can be roughly estimated by assuming a miss penalty. This miss penalty should take into account the just-mentioned features, which aim to reduce the leading-edge and trailing-edge effects. It should also consider the frequent data dependencies in the sampled application. Note that in both RS1 and Alpha, a functional unit pipeline stalls when an instruction requires the operand being returned on a cache miss.

Assuming the L2 cache satisfies all of the L1 misses, the Alpha L1 cache miss penalty is estimated at six processor cycles, 38 nsec at 160 MHz. At the fastest designed bus interface—half the processor clock—this is three 80-MHz Alpha bus cycles: request onto bus, L2 cache access, and data on bus.

The limited hit under miss may shorten the cache miss penalty by two cycles. However, the cache penalty may be lengthened by many factors such as data dependency, L2 and bus contention, and back-to-back misses. (Only the first miss obtains the hit-under-miss benefits.) For comparing the performance effects of the RS1 and Alpha caches, we simply chose six processor cycles as the cost for an Alpha cache miss. A pure comparison of cache designs would use the same six-cycle miss penalty for the RS1 design point. However, our exercise attempts to compare the effects of processor clock on performance and therefore includes effects of an L2 (in Alpha) to keep memory relatively close as the processor frequency is increased. For the moderate clock rate RS1, we used 10 processor (or RS1 bus) cycles (160 nsec) for L1 misses (satisfied by main memory). For each type of cache, the last row in Table 1 shows the estimated finite cache miss effect. We calculated it as total penalty cycles (total misses times miss penalty) divided by total instruction count.

L1 store-back penalties. In addition to the finite cache miss effect, the finite cache effect includes the penalties associated with writing stores to main memory. Alpha is somewhat unique among high-end RISC microprocessors in that it uses a store-through policy for the L1 D-cache; a four-entry (32 bytes per entry) write buffer reduces the store bus traffic.[4] Rather than delaying the effects of a store by modifying a cache line and then casting out changed lines as they are selected for replacement, stores modify the contents of the write buffer entries. At a later time, the processor transfers write buffer data to memory. Since the write buffer entries are the same length as an Alpha cache line, the four-entry write buffer provides a function similar to four RS1 store-back buffers.

Alpha does not support unaligned stores or stores of less than 4 bytes, so each of the 32-byte entries requires only 8 valid (and therefore dirty) bits. Each valid bit covers a cor-

responding 4-byte portion of an entry. When a buffer entry is emptied, the valid bits appear to the memory (or L2) as mask bits, which control the choice of 4-byte quantities to be written. Therefore, the original cache line data need not be fetched on a store miss. Alpha implements an allocate-on-read miss policy.

Until an entry transfers to memory, additional stores to the same cache line simply update the appropriate write-buffer entry. Head and tail pointers maintain a rough FIFO relationship among the four entries. The write buffer attempts to write its head entry to memory whenever the write buffer contains at least two valid entries—not when the buffer is full.

Table 2. Write-back bus traffic.

SPECint89 benchmark	No. of stores	Four 32-byte LRU write buffers	RS1 (store-in, 64 Kbytes, four-way, 128-byte line)
Gcc	3,322,758	1,390,366	12,176
Li	4,539,836	2,145,440	1,283
Eqntott	369,647	632	164
Espresso	1,318,309	224,940	3,064
Total lines written back	9,550,550	3,761,378 @ 4 cycles	16,687 @10 cycles
Write-back penalty cycles (per instruction)		0.125	0.001

Since the packaging of stores in the Alpha write buffer is not deterministic, a cycle-by-cycle Alpha simulation including caches would be required to accurately determine their effectiveness. We did not model this. We modeled four 32-byte write buffer entries assuming a least recently used (LRU) replacement algorithm. (This should underestimate the Alpha store traffic since the model does not purge an entry until the four-entry buffer overflows. Alpha, however, might purge an entry when there are only two valid entries.)

We counted the number of stores and the number of times the LRU buffer entry had to be emptied to make room for a new store request. Our previously mentioned cache simulations also provide cast-out counts, assuming store-in designs. Table 2 compares these results for the same SPECint89 traces we've described. The number of stores represents the store-through bus traffic without the write buffer. The effectiveness of the write buffer in minimizing store bus traffic is excellent (1/600) in the eqntott benchmark, fair (1/6) in espresso, and only marginal (1/2) for gcc and li. The write buffer reduces total store bus traffic for these samples (120×10^6 total instructions) by a factor of 2.5.

Alpha's small store-through cache and write buffer have significantly more store bus traffic than RS1's relatively large store-in cache. Determining the end-user perfonnance impact is difficult since the amount of penalty per bus transaction depends on the likelihood that the bus (or memory) is busy when the transaction is requested, as well as the resulting waiting period. Therefore, applications that have larger miss rates and the associated higher bus/memory use will experience larger cast-out penalties. The two main effects of the write buffer (compared to store-in) approach are the additional latency incurred by some stores (to a full write buffer) and the additional use of the memory subsystem. Increased use of the memory subsystem will increase the average penalty for cache misses.

For purposes of illustration, assume that a cast-out from the Alpha write buffer takes two bus (four processor) cycles. (The increase in L1 cache miss penalty, which results from the store traffic's effect on L2 use, is included in this cast-out penalty.) Table 2 shows a total of 3.76×10^6 cast-outs for Alpha; at four cycles each, this means 15×10^6 processor cycles.

For the RS1 design, the cast-out penalty is negligible when bus traffic is light, due to the store-back buffer. Although the miss and store activity is very light for the RS1 cases, for illustration we use a pessimistic value of 10 cycles per cast-out. The resulting RS1 penalty is two orders of magnitude smaller than the Alpha penalty estimate.

The finite cache effect for the four sets of SPECint89 samples is the sum of the finite cache miss effect (Table 1) and the store traffic penalty (write-back penalty cycles, Table 2). The Alpha finite cache effect would be 0.305 while the RS1 value would be 0.033—about a 9:1 ratio. The Alpha miss penalty estimate assumes the L1 miss being satisfied in six cycles; for systems with different L1 miss penalties, the total penalty cycles are approximately proportional. Two 200-MHz DEC Alpha systems with different off-chip memory subsystems provide an example of performance variance with an L1 miss penalty. The better memory subsystem gains about 5 percent in performance on the SPECint92 benchmark and 18 percent on SPECfp92.[11]

L2 caches. Alpha's aggressive clock rate requires a good second-level cache to achieve the assumed L1 miss penalty of six processor cycles. We assume that all L1 misses are satisfied by the L2 in Alpha (infinite L2 cache) and main memory in RS1. However, the introduction of an L2 adds the possibility of L2 misses. L2 misses are relatively expensive; for an aggressive clock rate design, the main memory is a significant number (20 to 30) of processor cycles away. Since the pipelines are likely to stall during an L2 miss, we can

The compound FMA instruction, along with the update forms of storage reference instructions, gives RS1 a clear path length advantage on floating-point applications.

estimate L2 miss effects by adding the total L2 miss penalty (cycles per L2 miss times the number of L2 misses) to the infinite L2 cache cycle count previously estimated. However, we make no estimate of the L2 miss counts or penalties.

To reduce the number of L2 misses, Alpha supports a prefetch into L2 instruction. Assume prefetch instructions are scheduled sufficiently far ahead of an L1 miss and sufficient memory-to-L2 bandwidth exists to sustain the transfers. Then, prefetching the "to be missed" data allows the L1 miss to be satisfied by the L2 without experiencing the additional penalty of an L2 miss. However, the performance impact of L2 misses may be significant in some applications. For example, large databases can be accessed in a semirandom (nonsequential) fashion. Since the databases are often orders of magnitude larger than affordable L2 capacities (or even main memory), the first access of each group of records is likely to result in an L2 miss. The randomness of the access patterns makes prefetching difficult.

A less obvious cause of L2 misses is a context switch. When many unrelated processes run in the interval between time slices for a given process, they may purge private data of that process from L1 and L2. Each time the process starts to run, it effectively has cold L1 and L2 caches. The L1 being purged is not as significant since the number of compulsory misses, and the resulting additional penalty, is small. However, due to its size, the number of cycles required to warm up the L2 may become a considerable portion of each time slice. (For a 1-Mbyte L2 with 64-byte lines and a 20-cycle miss penalty, the maximum additional penalty per time slice is 320,000 cycles. With time slice intervals on the order of a million cycles, this could be a significant overhead.) Therefore, a common assumption, that a data structure is established in L2 and the effects of L2 misses can be ignored, may not be valid under these conditions. In this particular situation, the use of the Alpha prefetch instruction to warm up the L2 is limited since neither the programmer nor a compiler could be expected to know when the time slice interrupts occur during any specific run of the program.

TLB effects. In addition to infinite cache CPI and finite cache effects, TLB misses contribute to the finite cache CPI.

RS1 has a 128-entry, two-way set-associative instruction TLB and a 128-entry, two-way set-associative data TLB. Each RS1 TLB entry maps one 4-Kbyte virtual page to the corresponding 4-Kbyte real page. RS1 hardware resolves TLB misses in roughly 40 to 50 cycles. Alpha partitions instruction TLBs into two types: eight fully-associative, 8-Kbyte-page entries and four fully associative, 4-Mbyte-page entries. The Alpha data TLB consists of 32 fully associative entries, each mapping an 8-Kbyte, 64-Kbyte, 256-Kbyte, or 4-Mbyte page. Alpha software resolves TLB misses.

Hardware TLB reloadings, large TLBs, and a compiler's data access pattern restructuring reduce the RS1 data TLB miss penalty on the SPECmark89 benchmark to negligible levels. The granularity supported for the Alpha data TLBs, and assumed exploitation by the operating system, provide adequate tools to make Alpha data TLB misses negligible on SPECmark89.

Instruction TLB misses are not negligible. We don't know many details of Alpha's four 4-Mbyte instruction TLB entries; they may be reserved for operating system use.[5] Since use of any of these entries requires blocking 4 Mbytes of contiguous real memory, and all 4 Mbytes must be paged in for the TLB entry to be valid, it does not seem likely that they would be allocated to a user task. If limited to only eight Alpha 8-Kbyte instruction TLB entries, significant miss rates can be expected on some applications. Using the instruction traces mentioned earlier, and an estimate of 100 cycles for a software TLB reload, simulations indicate about 25-30 percent degradation on the gcc benchmark. We may have understated this effect as the model assumes the eight instruction TLB entries are managed on an LRU basis. Alpha instruction TLBs are managed on a not-last-used basis.[4] Using the 50-cycle RS1 TLB miss penalty and the larger instruction TLB, simulations indicate 4 to 6 percent degradation for RS1 on gcc.

Overall CPI. Combining the pipeline effects from Part 1 with these cache and TLB effects, we conclude the following about CPI for the SPEC89 suite. The finite cache CPI should be significantly higher for Alpha than for RS1. On fixed-point codes, although the pipeline CPI appears to be similar for RS1 and Alpha, the differences in cache effects on these codes are significant. Simulations show that Alpha, even with a fast L2 cache to satisfy L1 misses, has significantly more total miss penalty due to the smaller, direct-mapped L1 caches. If no L2 is present, the total Alpha miss penalty will be significantly higher than our estimates. On floating-point codes, we expect the pipeline CPI to be significantly higher for Alpha than for RS1. Alpha's smaller, direct-mapped, store-through caches (with short lines) will further widen the gap in finite cache CPI values for the SPECfp89 benchmark.

Figure 1 shows how CPI scales with the clock rate. The lower curve illustrates constant path length and CPI—idealized linear scaling of performance with frequency. As clock rate increases to the limit bearable by the pipeline stages, designers can achieve higher clock rates by decreasing the amount of work per stage. The next curve shows CPI increasing with the clock rate; the CPI increases result from effects such as increased interlocks between functional units and longer latencies that result from lengthened pipelines. When comparing this pair of processors, note that the effects of lengthening the pipeline are more pronounced in floating-point codes than in fixed-point applications. The upper pair of curves reflect added CPI effects of the memory subsystems.

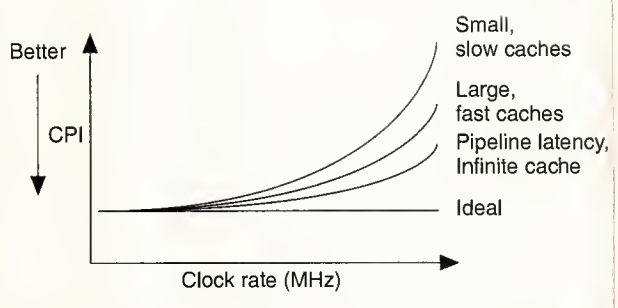


Figure 1. Effects of clock rate on CPI, including pipeline and memory subsystem.

Path length

We have presented the relationship between frequency and CPI in the context of a common instruction sequence for both RS1 and Alpha. Next, we examine the performance implications of some differences between the RS1 (POWER) and Alpha instruction sets. Instruction set architects consider hardware/software interaction when deciding how to partition functions between software and hardware. The instruction set itself may be influenced by the technology and cycle time expectations of the first implementation.

A subtle distinction exists between the clock rate and CPI differences previously discussed and the path length issues we discuss now. The former are implementation issues; the latter are architectural issues. Generally, designers determine implementation details, such as multiple functional units, pipeline lengths, and cache parameters, by the practical concerns of cost and speed. As technology changes, so do these features—even for a given architecture. Architectural issues, such as storage reference restrictions, special-purpose registers, and update forms of storage references, are much more difficult to alter as the software base grows. While the implementation details described earlier can change with each release of hardware, architecture tends to be more stable.

Path length is the number of instructions required to perform some function; it depends on the instruction set, the number of registers, and the compiler. Both machines have roughly the same number of registers. Since we are in no position to compare the compilers, we address the way the instruction set affects path length. In all cases, the function provided by one instruction in one architecture requires more than one instruction in the other. Obviously, the change in overall path length is a function of changes for specific sequences and their relative frequency in an application. Increased instruction path length does not proportionally decrease performance when the newly added instructions affect overall CPI.

Alpha's main path length advantage involves branches. Alpha branch instructions include some simple testing (odd, even, or compare to zero) of a general-purpose or floating-point register. Determining one of these attributes for a value in a register does not require an explicit compare instruction. RS1 requires a compare and a branch to replace Alpha's branch in such a case. The impact of the additional compare is highly dependent on the ability to schedule. If any other attribute must be determined (for example, being equal to the contents of another register or to an immediate value), Alpha also requires an explicit compare instruction. Alpha's test and branch capability provides no path length advantage in such cases.

Alpha also provides a pair of conditional move instructions, which allow a simple register test (odd, even, or compare to zero) to determine whether a specified register copy operation is performed. In the worst case, RS1 uses a three-instruction sequence (compare, conditional branch around copy, copy) to replace this one instruction. Again, when the user needs to compare two values to determine whether the copy is performed, Alpha also requires a compare. In this case, the benefit of a conditional move is the elimination of a forward conditional branch. The conditional move cannot be used to replace loop-closing branches, subroutine calls, and goto constructs. For some of the remaining RS1 branches, the associated compare is scheduled so that the branch is resolved without guessing. Some unresolved branches are predicted correctly, incurring no penalty; the remaining branches incur a one- to three-cycle penalty. By removing one, two, or three penalty cycles on some branches, Alpha should obtain a minor gain on general codes. A slight additional improvement occurs in cases where Alpha does not require the explicit compare.

Several RS1 instructions do not exist in Alpha, one class of which is storage reference instructions that provide an implicit update of the address pointer. Rather than counting them as a path length advantage, we considered the pipeline effects of these implicit updates in the pipeline CPI effects by giving RS1 credit for a "virtual" functional unit. However, the update forms also provide some I-cache footprint reduction.

A second class of instructions is load multiple and store

The distinguishing performance characteristic of the Alpha systems is clock rate; the distinguishing characteristic of the RS1 is support for instruction-level parallelism.

multiple—used primarily in subroutine prolog and epilog. Although implementations of these instructions could move multiple registers each cycle, RS1 moves one register (4 bytes) per cycle. Their strength comes from a single instruction replacing a moderate number of instructions. The resulting reduction in footprint increases the effectiveness of instruction caches, instruction buffers, and instruction issue bandwidth. Since a pair of these instructions replaces up to 64 load and store instructions, code density improvement can be significant.

The RS1 string operations provide similar I-cache benefits to the load multiple and store multiple instructions. Additionally, string operations allow block moves with no alignment requirements; RS1 handles all string operations in hardware.

The RS1 branch on count instruction replaces a loop-count decrement and a branch. Since its predominant use is in numerical applications, and it saves only one fixed-point unit instruction per loop iteration, its advantage will be minimized by the likely unrolling of floating-point loops for Alpha.

For floating-point code sequences, a major path length advantage of RS1 is the floating-point multiply-add (FMA) instruction. FMA instructions account for a significant percentage of the RS1 floating-point unit operations in SPECfp89, Linpack, Livermore loops, and many scientific/engineering applications. The compound multiply-add noticeably benefits some subset of floating-point codes.[12] Alpha requires two six-cycle floating-point unit pipeline passes to provide the function of the RS1 FMA (A = B * C + D) operation. Since the requirement for two passes stems from the need to use two instructions (multiply and add) to replace the compound instruction, this is a path length issue. Replacing the single FMA with a pair of dependent floating-point unit instructions does not increase the instruction-level parallelism. Therefore, if the FMA is on the critical path, doubling the path length of this compound operation doubles the corresponding cycle count.

The compound FMA instruction, along with the update forms of storage reference instructions, gives RS1 a clear path length advantage on floating-point applications.

Alpha architects rely heavily on compiler support to select instruction sequences that avoid performance pitfalls exposed by hardware simplifications. For example, Alpha's storage references can be grouped into two categories. Four-byte (8-byte) references that do not cross 4-byte (8-byte) boundaries can use normal loads and stores without penalty. For all unaligned references, the Alpha programmer has two choices. Using normal loads and stores is very slow, possibly generating warning messages to the user.[6] The suggested alternative is special load-unaligned instructions that guarantee alignment by forcing the low-order address bits to zero. References to two adjacent elements are required to obtain unaligned 4-byte and 8-byte items. Additional instructions are required to extract or merge byte or 2-byte quantities. The compounding effect of alignment restrictions and lack of byte and 2-byte references is illustrated by the various instruction sequences shown in Alpha literature.[6] The compiler must attempt to select the most efficient sequence. The best choice depends on the particular alignment, if known at compile time, and the storage reference access size.

In contrast to requiring instruction sequences for alignment and byte manipulation, RS1 loads (stores) often can be used without considering alignment. Frequently encountered types of unaligned references (unaligned 4-byte integer operands or 8-byte floating-point operands aligned on odd 4-byte boundaries) incur only one penalty cycle on RS1. Therefore, the path length of compiled code for these cases is a single instruction on RS1 whereas Alpha requires a sequence.

Evaluating performance

Vendors have reported SPEC Release 1.2 performance on many systems.[13] Figure 2 shows some of these results, including labels that indicate the underlying architecture. (Since June 1992, SPEC has published higher results for many vendors. However, we don't intend Figure 2 to be an exhaustive set of competitive data. Data points are selected to illustrate the lack of correlation between clock rate and performance.)

Although performance tends to scale with clock rate for a given architecture, no clear relationship between clock rate and performance exists across architectures. The range of the SPECmark89 results at a given frequency, as well as the various on-chip frequencies required to deliver a given SPECmark89 result, clearly indicate that many factors other than frequency contribute significantly to performance.

Although we've discussed the magnitude of the performance impact for a few of the items in the two compared machines, we did not attempt to project overall performance. However, the overall impact of the performance effects we've

described can be demonstrated by the measurement data.

Table 3 lists the results for the two systems on various processor benchmarks.[14-16] On SPECint92, the 160-MHz Alpha leads the 62.5-MHz RS1 by 50 percent; on the remaining benchmarks, the systems differ by less than 10 percent. As an average across these benchmarks, the RS1 performs at about 90 percent of an Alpha yet requires only 40 percent of the clock rate. Instruction-level parallelism is a practical alternative to aggressive clock rate for achieving performance.

While this Alpha system runs at more than two and one-half times the clock rate of the RS1, performance is comparable. The distinguishing performance characteristic of the Alpha systems is clock rate; the distinguishing characteristic of the RS1 is support for instruction-level parallelism. This difference in philosophies will become more pronounced as DEC continues to increase clock speed and IBM enhances superscalar capabilities, supporting multiple functional units, both fixed-point and floating-point. IBM PowerPC chips will merge the aggressive clock rate and superscalar approaches.

WE HAVE SELECTED ONE PROCESSOR to represent design opportunity available at 62.5 MHz and another to represent a 160-MHz design point. Based on the information currently available, we compared the performance aspects of the functional units, the interaction of the functional units, and the memory subsystem effects of these two processors.

We attempted to identify those details that were likely to be influenced by the designer's clock rate goal. Reviewing the trade-offs for the two processors showed that higher clocks are associated with longer pipelines; less instruction and operand queueing; less autonomy of the fixed-point and floating-point units; and smaller, simpler caches. The aggressive clock rate goal is also evident in the instruction set definition. The Alpha architecture document describes areas in which it relies on instruction sequences to replace single instructions that might be difficult to implement at higher clock rates, such as unaligned or byte storage references.

The major advantage for Alpha is its higher clock rate. A moderate CPI advantage of Alpha results from separate ALU and load and store units. However, this benefit is offset by dispatch restrictions, the virtual dual fixed-point unit support provided by RS1 update forms, and Alpha's longer load use delays. A moderate path length advantage for Alpha stems from branches that incorporate simple tests, which, in some cases, remove the need for an explicit compare instruction.

The major CPI advantages of RS1 come from its FMA support; shorter pipelines; and larger, set-associative, store-in caches. Moderate CPI advantages of the RS1 come from early branch resolution, more flexible dispatch and issue, fixed-/floating-point unit instruction queues, and the pending store queue. RS1 gained moderate path length advantage with load multiple and store multiple instructions, which favorably affect the I-cache footprint.

In addition to clock rate, many factors contribute to measurable end-user performance and may be negatively affected by the clock rate and cost goals. The historical lack of correlation between clock rate and performance is best illustrated by a plot of SPECmark89 versus processor clock rate.

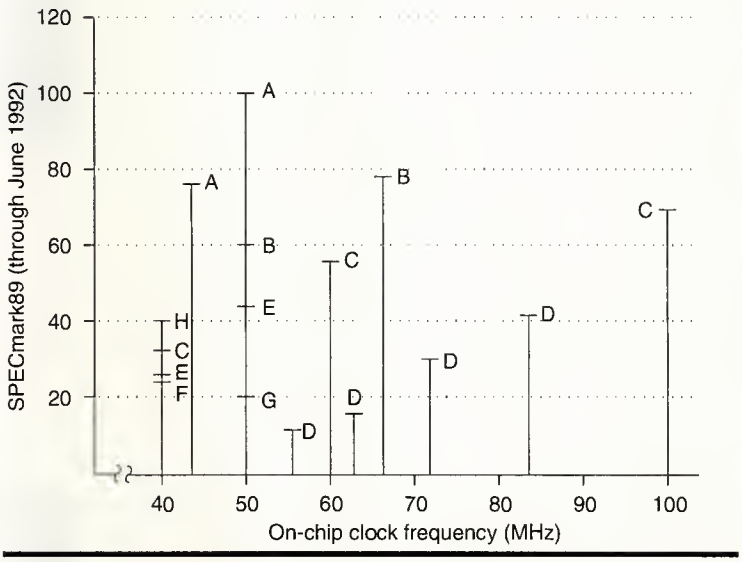


Figure 2. Cycle time is not enough to project performance. (Letters A-H indicate different instruction set architectures.)

Table 3. Benchmark measurements.

Benchmark	RS1 (62.5 MHz)	Alpha (160 MHz)	Ratio
SPECmark89	126.4	137.3	1.088
SPECint92	61.9	94.6	1.528
SPECfp92	134.6	137.6	1.022
Linpack(100×100)	38	36	0.947
Linpack TPP	104	114	1.096

We plan to continue evaluating the performance of industry-leading systems. Assuming sufficient reader interest, future articles will address follow-on systems.

Acknowledgments

We thank the processor and system designers at DEC and IBM for providing interesting design points to analyze. We also thank Sohel Saiyed for generating the traces, running the cache model, and modeling the Alpha write buffers; Juho Tang for modification of the cache model; and Rich Oehler, Kaivalya Dixit, Chuck Moore, and John Reysa for reviewing the article and providing many useful comments.

This article was submitted August 20, 1992, and revised for publication on August 20, 1993.

References

1. "IBM RISC System/6000 Technology," Order no. SA23-2619, IBM Corp. branch offices, 1990.
2. *IBM J. Research and Development,* Vol. 34, No. 1, Jan. 1990, entire issue.
3. *AIX Version 3.2 for RISC System/6000 Assembler Language Reference Manual,* Order no. SA23-2197, IBM Corp. branch offices, 1992.
4. "DECChip 21064-AA RISC Microprocessor Preliminary Data Sheet," Digital Equipment Corp., Maynard, Mass., Apr. 29, 1992.
5. Richard Sites and Richard Witek, "Alpha Architecture and First Implementation," Proc. Compcon, IEEE Computer Society Press, Los Alamitos, Calif., Feb. 27, 1992, p. 214.
6. *Alpha Architecture Handbook,* Digital Equipment Corp., 1992.
7. Daniel Dobberpuhl et al., "A 200-MHz, 64b Dual-Issue CMOS Microprocessor," *Proc. Int'l Solid-State Circuits Conf.,* IEEE CS Press, Feb. 20, 1992, pp. 106-107.
8. "DEC Enters Microprocessor Business with Alpha," *Microprocessor Report,* Mar. 4, 1992, pp. 1 and 6-14.
9. David A. Patterson and John L. Hennessy, *Computer Architecture: A Quantitative Approach,* Morgan Kaufmann Publishers Inc., San Mateo, Calif., 1990, p. 422.
10. "SPEC Benchmark Suite Release 1.0," *SPEC Newsletter,* Spring 1990, Vol. 2, No. 2, pp. 3-4.
11. *SPEC Newsletter,* Vol. 5, No. 2, June 1993, pp. 25, 28, 51, 54.
12. "IBM RS/6000's Complex Implementation Extracts Peak Performance," *Microprocessor Report,* Aug. 21, 1991, p. 14.
13. *SPEC Newsletter,* issues through June 1992.
14. Jack J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software," CS-89-85, Apr. 15, 1993, p. 6. (Postscript copies available through netlib@ornl.gov; in message, type "Send performance from benchmark.")
15. *SPEC Newsletter,* Vol. 4, No. 4, Dec. 1992, p. 62.
16. *SPEC Newsletter,* Vol. 5, No. 2, June 1993, pp. 26, 34, 52, 60, and 146.

Steven W. White is a senior engineer in IBM's AWS Processor Architecture and Performance Group. His primary interest is scientific and engineering performance. He joined IBM to work on mainframe scientific and engineering processor development, architecture, and system design.

White received his BSEE, MSEE, and PhD degrees from Texas A&M University, where he also taught for three years. He has published papers in a variety of areas including VLSI design, parallel processing, numerical algorithms, code optimization, system design, performance analysis, human genome research, and compilers. He is a member of the IEEE and the IEEE Computer Society and is a registered professional engineer.

Phil D. Hester is vice president, systems and technology for Advanced Workstations & Systems (AWS) in Austin and a member of the AWS Business Council. He holds responsibility for advanced systems and technology development of IBM RISC System/6000 products. Previously, he was responsible for development of the RS/6000 family and has been involved in the architecture, performance analysis, design, and implementation of the RISC-based products in IBM.

Hester received a BS degree in electrical engineering and an MS degree in engineering from the University of Texas at Austin. He is a member of Tau Beta Pi and Eta Kappa Nu, holds numerous patents, and has authored technical publications on RISC processors.

Jack W. Kemp, manager of Hardware Architecture, Advanced Workstations and Systems, is currently responsible for both processor and I/O architecture for the RISC System/6000 family of products. He managed the SCSI subsystem development effort for the original RISC System/6000 product and has held responsibility for processor architecture and performance.

Kemp holds a BS in solid-state physics from Marist College in Poughkeepsie, New York.

G. Jeanette McWilliams is an independent consultant who specializes in processor and memory subsystem performance. Previously, she worked at IBM as a senior programmer and hardware performance analyst on the RS/6000 team. She projected RS1 performance on industry-standard benchmarks prior to chip fabrication. She was IBM's first technical representative to SPEC and served on the Steering Committee for two years. Other performance assignments at IBM include transaction processing system support on mainframes and PC software performance assurance.

McWilliams received a BA degree in mathematics and an MA degree in computer science from the University of Texas at Austin and has completed IBM's Systems Research Institute. She is a member of the Association of Computing Machinery and the Independent Computer Consultants Association.

Direct questions concerning this article to Steven W. White, ZIP 9440, IBM Advanced Workstations and Systems, 11400 Burnet Road, Austin, TX 78758; or steve@hwperform.austin.ibm.com. McWilliams may be reached at wk00196@worldlink.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 162 Medium 163 High 164

Software Report

Cooperation: Japan's new watchword?

David K. Kahaner

US Office of Naval Research, Far East

kahaner@cs.titech.ac.jp

According to a report recently released by Japan's Management and Coordination Agency, that country spent over 13 trillion yen (US$100 billion) on research and development in 1990, 10.7 percent above the previous year. Japan's research and development expenditures accounted for 3 percent of its GNP, a new record. The private sector accounted for about 11.7 trillion yen, 82 percent of overall R&D expenditures.

Reflecting the growing awareness of the importance of environmental protection worldwide, expenditures for environmental protection surged 20.3 percent over the previous year to 237.8 billion yen. Japan's exports of technology (receipts from patents and royalties) increased 3 percent over the previous year to 339.4 billion yen. Imports of technologies increased to 372 billion yen, 12.7 percent over the previous year. Japan's imports of technology from the US outpaced exports by 2.5 times.

As Japan attempts to keep pace with developments in the ever-changing world of high technology and computer science, its government and industry leaders are moving on several fronts simultaneously. As I describe in the snapshots that follow, efforts are under way to improve coordination and cooperation between the public and private sector there, as well as with foreign entities. Indeed, cooperation as much as competition appears to be the watchword in many of these efforts—cooperation between government agencies and private industry, with foreign businesses on either side of the Pacific, and between governments.

Stronger regional technical centers

The Ministry of International Trade and Industry's Agency of Industrial Science and Technology (AIST) plans to develop new regional technology policies and strengthen the research functions of the seven regional Government Industrial Research Institutes (GIRI) under AIST's umbrella in fiscal year 1993. The policies are designed to correct the over-centralization of industry. At the same time, upgrading unique R&D bases should prove indispensable to invigorating the regions. AIST aims to actively develop the various regional research resources centering on the GIRIs, marshal the public testing and research institutes (*kohsetsushi*), third-sector research centers, and private companies to play a guiding role in the regional development, and link them with local universities.

The importance of exchanges between GIRIs and *kohsetsushi* centers has been stressed before, but the various players have not yet conducted exchanges in a sustained fashion. AIST will support research by arranging tie-ups between *kohsetsushi* centers, private companies, universities, and other organizations. Changing *kohsetsushi* centers into industrial technology centers and renovating facilities and systems is moving ahead. By linking the centers to local universities, the policy will open the way for researchers at GIRIs to earn doctorates based on research achievements there.

ASIC chip makers mount challenge

Japan's semiconductor makers, faced with a mounting offensive on their home turf by US companies, are hustling to build up a presence in application-specific integrated circuits, in some cases by cooperating with US companies. ASIC chips have come into their own as semiconductor users move away from multipurpose chips, but they also have assumed heightened importance in light of the market penetration by US

0272-1732/93/1000-0090$03.00 © 1993 IEEE

chip makers.

Demand is growing for programmable logic devices and field-programmable gate arrays from system developers looking for devices to incorporate high-speed systems that outstrip the capacity of conventional gate arrays. The programmable logic market amounted to US$900 million in 1991, only around 25 percent of the size of the gate array market. According to industry estimates it should expand to around US$2.2 billion by 1995.

Disc makers swap patents

Hoping to avoid another bitter confrontation over two competing technologies, Matsushita Electric Industrial Co., Sony Corp., and Philips Electronic BV reportedly have concluded basic agreements to make their patents on digital compact cassettes and minidiscs mutually accessible. The two Japanese manufacturers have led rivals in the commercialization of the technologies—small digital recording devices that generate high-quality sound. Matsushita adopted the cassette format, while Sony picked the disc format. Philips has committed itself to both types of equipment.

Matsushita, Sony, and Philips will control their patents uniformly, supplying technology to domestic and overseas enterprises interested in producing both hardware and software for digital compact cassettes and minidiscs. Mutual access to the patents on compact cassettes and minidiscs will allow the two Japanese camps to hone technology applied to their products, company officials say. Meanwhile, the agreement will pave the way for Matsushita to penetrate Sony's turf and vice versa. Matsushita has begun to market its first digital compact cassette recording, while Sony released its minidisc recorder at the end of 1992.

Construction industry discovers satellites

Japan's construction industry has joined the growing contingent of businesses worldwide that have turned to the heavens for guidance—not to the stars, mind you, but to the satellites that make up the Global Positioning System (GPS). The GPS is a group of satellites launched by the US Department of Defense that was originally intended for boat and aircraft navigation. At present, 18 GPS satellites orbit 20,000 km above earth. Three more are scheduled for launch by the end of 1993, bringing the total to 21, or enough to provide 24-hour access anywhere in the world. While car navigation systems form the highest profile commercial GPS application, the construction industry is learning to make use of the GPS satellites for land survey work as well.

Surveying is simple with GPS—all you need is a pair of receivers, antennas, and three satellite signals. Measurements of the distance between the two receivers and the elevation at the site of second receiver can reach an accuracy of 0.0001 percent. The GPS system eliminates the traditional equipment surveyors use to measure distance and angles.

The first commercial GPS system came out in 1986, and already this method has become the standard tool for many survey applications in the US. Europe was also quick to make use of the system in 1987, but Japan has lagged a bit. Of the roughly 40,000 GPS survey receivers that have sold worldwide, only 250 reportedly are in operation in Japan. However, several surveying firms there are now using the system, and more are likely to follow.

"The domestic market for GPS survey receivers finally began to take off last year," says Hideyuki Torimoto of Trimble Navigation Systems, an American firm that leads the world in the manufacture of GPS receivers. As recently as two years ago, GPS receiver sets cost as much as 15 million yen ($120,000). Trimble now markets a model for general precision survey work that costs only 5.9 million yen.

Japan, South Korea, Europe cooperate on fast ISDN

These traditional competitors agreed recently to jointly develop a large-scale communications network that would transmit data among countries at high speed, according to government officials in both countries. The accord came during a regular conference between the Japanese Posts and Telecommunications Ministry (MPT) and its South Korean counterpart in Seoul. The integrated service digital network (ISDN) would permit transmission of different communications services, including digital telephone calls, facsimile transmission, and data communications, on a single network.

Many countries now are studying the possible commercial operation of the new system. However, because ISDN system transmission modes vary from nation to nation, some countries have found it impossible to exchange data and information. To standardize ISDN, Japan and the European community are scheduled to experimentally connect their communications lines next year. This agreement will help to standardize communications modes in the Asia-Pacific basin. To promote their joint project, Japan and South Korea soon are likely to select the participating communications enterprises and communications equipment manufacturers.

Computer downsizing trends

Japanese computer makers, which have long challenged IBM's supremacy in mainframes, are now following that company's lead in the opposite direction. Like IBM, they are reducing their dependence on large computers in favor of system planning and maintenance services. But the pace of the shift away from big, costly computers to a network of smaller machines has been slow compared with the US. Here, the downsizing trend has been so rapid that IBM has predicted zero growth in hardware revenue and other firms have abandoned the field altogether.

The story is different in Japan. There, though down from 70 percent in 1989, mainframes still hold 60 percent of the market, and smaller machines are suffering from the sluggish economy. Yet many observers say that change is inevitable. The limited availability of Japanese language software for networks of small computers has slowed the pace, they say. Also, the greater variety of computers sold in Japan, making networking more difficult, has impeded progress. Industry estimates put the percentage of personal computers and workstations forming networks at less than 10 percent in Japan, whereas the number is reportedly between 30 and 40 percent in the US.

According to Shozo Shigeoka, editor-in-chief of *Nikkei*, a leading computer magazine, "department heads of many Japanese companies make decisions after discussing subjects with all bosses up the ladder. Distributed computing does not fit well with this centralized decision-making style." Everybody wants to look at all the information, he says, while lower ranking western managers have more decision-making power. That, combined with the substantial mainframe software assets they have accumulated, has made Japanese companies less eager to try new systems, according to Shigeoka.

IBM seeks Canon's help

IBM has added its personal computer operations to the growing list of areas in which it has sought the help of a prominent Japanese manufacturer. IBM and Canon have agreed to cooperate in the development of desktop and portable computers. This news coincided with an announcement by IBM that its PC development, manufacturing, distribution, and marketing operations would be consolidated in a new, autonomous unit known as the IBM Personal Computer Company.

A Canon spokesman says that one of the first tasks of the new alliance will be to develop a portable PC with a built-in printer. The partnership will help IBM tap Canon's expertise in computer peripheral equipment, especially printers. IBM already has joined forces with Toshiba and Hitachi to develop advanced semiconductor chips and high-end printers. Canon possesses technology in color flat-panel displays and optical magnetic disks that also could be of interest to IBM.

Scientific research is growing outside Tokyo

Scientific and technological activities in regions outside Tokyo are increasing, according to a recent report from the Science and Technology Agency (STA). That organization found that research institutes, led by those in the private sector, are moving increasingly into regions outside of Tokyo and its three surrounding prefectures, Kanagawa, Saitama, and Chiba. The report also called for harmonizing national and local science and technology policies, increasing the communication between the national and local governments, and improving the quality of regional science and technology.

According to the STA white paper, corporations in the private sector are accelerating regional scientific and technological activities as they shift their research centers to outside the metropolitan region. The report noted that research institutes of private firms and public organizations have been moving into regions outside the Kanto area at a considerable pace for the past several years. They tend to concentrate in a limited number of areas near major cities in each prefecture.

Noting that in recent years the private sector has established very few sizable research facilities in Tokyo, the report says that the Kanto area saw its share of newly opened research institutes sharply reduced to 34 percent for the 1989-1991 period, down from 52 percent in the previous three-year period.

Touching on growing international cooperation in scientific and technological research, the report says, "it is possible for local authorities to join hands with the national government" in carving out policies in some fields. It cites the difficulty of securing personnel and providing an "adequate living environment" for researchers in regions outside Tokyo. The report notes the need to improve the quality of regional science and technology, as well as to work jointly on policies and facilitate communication between the national and local individuality, and comprehensive policy-making.

"The national government cannot dishearten regional independence," the report concludes. "Our work suggests that local government develop advanced science and technology, adding national policies to regional potentials."

A Canon spokesman says that one of the first tasks will be to develop a portable PC with a built-in printer.

New Products

Joe Hootman

University of North Dakota

Communications/displays

Communicator combines functions

Developed jointly by AT&T and Eo, Inc., the 440 Personal Communicator combines fax, electronic mail, personal productivity applications, and cellular phone capabilities. Available at AT&T Phone Center Stores, this handheld, pen-based device comes in a variety of configurations, including one equipped with an 8-Mbyte RAM, 20-Mbyte hard drive, and internal modem. *AT&T; from $1,999.*

Reader Service No. 10

Power-stingy fax modems

PCMCIA-based fax modems offer battery-saving ultra-low-power mode, flash memory for on-line operating program upgrade, and built-in DAA line interface circuits. Operating at 50 mA and dropping below 2 mA in an on-command sleep mode, the SmartExchange 1414 provides data and facsimile capabilities, error correction, and data compression in a credit-card-size format. Offering full-duplex data communication at 14.4 Kbps, the 1414 permits four-to-one data compression, giving an effective throughput of up to 57.6 Kbps.

Entry-level 9624 and 9624E models operate at 2.4 Kbps for full-duplex communications; both come with standby and sleep modes. The "E" version accommodates built-in high-level error correction and data compression. *Smart Modular Technologies; $549.99 (1414), $399.99 (9624E), $299.99 (9624).*

Reader Service No. 11

Ethernet LAN, Token Ring adapter cards

Sixteen-bit I/O map ISA bus Ethernet cards boost throughput by using I/O and memory mapping modes to issue early interruptions so packets transfer quickly to host computer memory. Built for PC/XT/AT bus ISA-compatible systems, these LAN adapters work well with Novell's NE2000 Ethernet card and comply with Ethernet 802.3. The jumperless CN888E lets users fine-tune performance by configuring the I/O address, IRQ channel, boot ROM, fast read, CHRDY control, and I/O16 control. Both CN200E and CN600E come with a 16-Kbyte data packet buffer and onboard 8-Kbyte PROM socket for adding a remote-boot ROM.

Also available for 16-bit performance is the CN2000T Token Ring adapter. Supporting 4- or 16-Mbps network speed and offering both 8- and 16-bit data transfer, this AT- and ISA-system compatible card accommodates either STP or UTP wiring. *CNet Technology; $299 (CN888E), $99 (CN600E, CN200E), $439 (CN2000T).*

Reader Service No. 12

CNet CN2000T Token Ring adapter

LIU cuts digital transmission costs

Functionally compatible with most industry-standard single-channel transceivers, the VP14Q574 line interface unit gives users power dissipation advantages compared to discrete T1/E1 LIUs. Needing a single external crystal, it supports both DSX-1 (ANSI) or E1/CEPT (CCITT) formats for channel banks, multiplexers, office repeater relays, digital cross-connection systems, and digital switches. An 8-bit microprocessor interface, common to all four transceivers, provides extensive monitoring and control capabili-

0272-1732/93/1000-0093$3.00 © 1993 IEEE

ties. Packaged in a 128-pin plastic QFP, its features include group selection of jitter attenuation direction and internal multiplexers for nonintrusive monitoring of eight transmit/receive data paths. *VLSI Technology; $42 (10,000s).*

Reader Service No. 13

Handheld protocol analyzer

A PC-based protocol analyzer for notebook computers, the ParaScope 64M operates as a stand-alone device that does not require a card slot. Communicating via standard 4- or 8-bit parallel ports, this compact device running Feline software handles a wide range of protocols, achieving a throughput rate of better than 64 Kbps on a 25-MHz 386 machine. Available data codes include DDCMP, CRC-CCITT, CRC-16, CR-12, CRC-6, LRC, and parity error checking. Packaged in a 9.22×4.76×1.97-inch molded plastic case, PS6145 works off both AC and battery power, and comes with a battery-saving low-power mode. *Frederick Engineering; $3,295.*

Reader Service No. 14

Multiline voice, fax processing

Voice Ranger, a multiline voice processing board, features two ports per card and permits expansion to 16 ports per system. Using the device's 16-Kbits of memory per card, users can create multitasking voice applications under DOS, directly record voice files, and simultaneously record incoming speech on multiple lines. With the complementary Commando Developer's Tool Kit, a multitasking TSR operating in the DOS environment, developers can create fax mail, audiotex, voice bulletin boards, fax-on-demand, and order entry with credit card authorization applications.

V/S Plus, a multiline voice and fax processing package, includes a multiline voice card, plus voice- and fax-processing software. Offering fax-on-demand with most CAS-compatible fax cards, V/S Plus accommodates multilevel call processing applications and 1,000 voice mail boxes. Also, Faxmouth, a fax-on-demand add-on to the Bigmouth single-line voice processing system, lets callers leave voice mail messages and request faxed documents in a single phone call, retrieve stored documents, and fax them to selected fax numbers. *Talking Technology; $599 (Voice Ranger), $169 (Commando), $699 (V/S Plus), $199 (Faxmouth).*

Reader Service No. 15

Tool kit for IBM LAN systems

The NDIS Driver Developer's Tool Kit gives programmers samples, specifications, test tools, and documentation needed to design and implement NDIS 2.0.1 media access controller (MAC) device drivers. Features include an NIF validation program, NDIS trace tool, and IBM Token Ring sample driver. *DWB Associates; $575.*

Reader Service No. 16

RS-422, RS-485 support

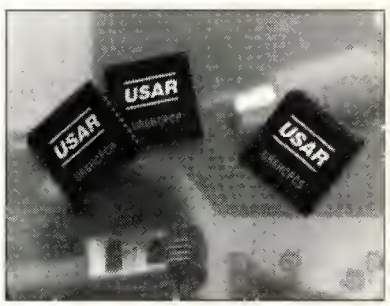
For greater flexibility in implementing advanced, client-server systems without relying on Ethernet-based expansion, RS-422 and RS-485 support has been added for these standard SBus expansion boards. With RS-422, Sparc systems connect to dozens of terminals, modem banks, and other peripherals at up to 4,000 feet. The multidrop capability of RS-485 works well for applications requiring numerous peripheral connections along a single serial line. Compatible with CCITT V.10/11 and X.26/27, the four-port module meets EIA standards. *Aurora Technologies; $499.*

Reader Service No. 17

Protocol converters

Providing up to 9,600-bps serial transmission speed, the single-chip UR6HCPCX HCMOS protocol converter links PC-compatible user input devices to hosts having a serial or parallel port. By providing all electrical and functional features of an 8042 system keyboard controller, plus all tiers of keyboard BIOS control, it matches performance formerly obtained through AT/PS2 motherboard connections. Housed in 40-pin DIP, 44-pin PLCC, or QFP packages, the converter presents data to the host in either PC-standard scan code format or translated extended ASCII codes. *USAR Systems; $27.50 (100s).*

Reader Service No. 18



USAR UR6HCPCX protocol converters

QUICC chip controller card

Dubbed the VCOM-54, this VMEbus communication controller uses Motorola's 32-bit 68360 Quad Integrated Communications Controller to provide four or eight ports of T1 or E1 speed connectivity through its P2 or front panel connectors. The controller features 1 or 4 Mbytes of parity-protected DRAM, VME 32/64-bit compatibility, VME DMA, and 68020 software compatibility. *SBE; from $2,385 (100s).*

Reader Service No. 19

Token Ring hub

An intelligent 16-port Token Ring hub, the TokenEase Smart MSAU lets network managers control ring-in/ring-out loopback, monitor port activity status, set password security, print reports, and view the port connection status via an on-screen worksheet. Controlled from a remote PC via out-of-band software management and through an RS-232 port, TokenEase features fault-tolerant ring-in and ring-out in case the trunk connection is broken, auto-loop back for preserving ring integrity, and LED diagnostic indicators

for troubleshooting. Included software can control 256 daisy-chained units and features a scenario function for downloading multiple network configurations. *MUX Lab; $1,750.*

Reader Service No. 20

Compact Token Ring MAU

Designed to fit a standard 19-inch rack, but only 1-inch deep, the 8228-equivalent Compact MAU fits smaller wiring closets or in-office locations. With eight lobe ports plus ring-in/ring-out, the unit's global reset button simultaneously initializes all lobe ports. Adjacent ring-in/ring out ports enable cleaner rack-style cabling. *Belkin Components; $499 discounts available).*

Reader Service No. 21

Switchmode control IC

Operating directly from telephone line voltages, the Si9114 enables operation at switching frequencies beyond 500 kHz, allowing for use of smaller magnetics and filters, and eliminating the need for electrolytic capacitors. Featuring soft-start, internal start-up, and latched shutdown circuitry, the 14-pin device can implement either a flyback or a forward converter at switching frequencies of 1 MHz. Available in plastic DIP and SOIC packages, its synch output pin allows additional power converters synchronized to each other or to an external clock, both in phase and frequency, simplifying EMI filtering.

Planar transformer techniques reduce external parts count and circuit board area, making for an 8-mm board thickness. *Silconix; $1.73 (OEM quantities).*

Reader Service No. 22

Color flat panel display

Designed for use in industrial controls, test and measurement instruments, and medical equipment, this EL display offers wide-angle viewing and a wide operating temperature range. Displaying eight colors on a high-contrast black background, the EL640.350-DA1 uses a nonreflective structure that needs no contrast-enhancing filters. For easy mounting, the EL glass panel and control electronics are assembled into a 226×153×20-mm package. Typical power consumption is 16W. *Planar Systems, Inc.; $2,495 (each), $1,210 (1,000s).*

Reader Service No. 23

Planar Systems' EL640.350-DA1

Touch screen controller

Packaged in a plastic box for mounting on the back or bottom of a CRT, this CMOS touch screen controller simplifies monitor refitting by avoiding internal mounting complications. A pocket-sized 3.75×2.5×0.9 inches, the SMT-1 operates off 70 mA of current and requires a single +5V power supply (or from any voltage from +8 to +16 VDC). The controller achieves a MIL-HBK-217-F1 MTBF rate of 572,600 hours. *MicroTouch Systems, Inc.; $318 (volume discounts available).*

Reader Service No. 24

Low-radiation monitors

Offering 0.28-mm dot pitch and 1,280×1,024-pixel resolution, the ASTVision low-radiation, multi-sync color monitors come in 14-, 15-, and 17-inch formats. Push-button, digital controls allow for horizontal and vertical sizing and positioning. Featuring 10 preset or eight custom modes for changing display configurations in a Windows-based environment, the monitors come with recyclable bromide-free plastic cases. *AST Research; $495 (14-inch), $595 (15-inch), $995 (17-inch).*

Reader Service No. 25

DSP components

Parallel processing boards

Two DSP boards provide support for eight 50-MHz TMS320C40 DSPs and up to 400 Mflops in a single PC or VME slot. Featuring a scalable topology that enables addition of TIM-40 standard single and twin processor modules and boards, C40 Octal Processor Systems can be used to build large networks for use in advanced imaging, sonar, radar, simulation, and modeling system applications. Offering direct host access to each processor through the JTAG interface, the systems allow code debugging in C, assembler, or both simultaneously. Accompanying software tools include an assembler/linker, ANSI C compiler, C source-level debugger, 3L parallel C compiler, Virtuoso, and FloTar and FasTar libraries. *Spectrum Signal Processing; $19,145 (PC), $18,240 (VME), volume discounts available.*

Reader Service No. 26

Multi-DSP board for VMEbus

Well suited for high-speed results in image processing, radar and sonar signal analysis, telemetry and telecommunications, simulation, and aerospace applications, the DSP-4 board moves data at over 1.2 Gbytes/s and processes at more than 300 Mflops. Built around three TMS320C40 processors, each board boasts 1 Mbyte of no-wait private RAM space, expandable to 16 Mbytes, and a 4-Mbyte block of shared memory available to all on-board processors. Included are the SPOX library of DSP routines, with FFT, other array and transform operations, FIR and IIR filters, and transcendental functions. Requiring 60W of 5V power, the DSP-4 fits a single 6U slot of a standard VMEbus card cage. *Pacific Cyber/Metrix: $15,349.*

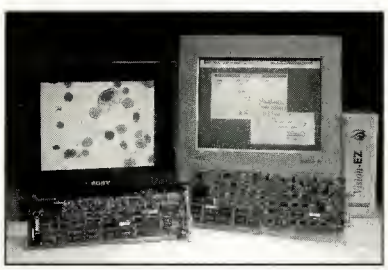
Reader Service No. 27

Data acquisition boards

Offering on-board simultaneous sample-and-hold and DSP intercon-

nection, FAST Series boards operate at a 1-MHz peak channel sample rate and 250 kHz/channel across four channels. Suited for applications requiring measurement of multiple sources of simultaneous events such as automated test fixtures and medical data acquisition, the SSH version features four channels of SSH resident in a single PC/AT slot, enabling implementation of DAS applications that require 12-bit resolution. The DSP version comes in either DSP-Link or DT-Connect; both feature a daughterboard containing a 4-Ksample FIFO buffer. The SSH version can also include the DSP option. Vibration analysis, speech processing, and spectral analysis are typical uses that require high-performance data acquisition coupled with DSP for real-time data reduction and analysis. *Analogic; from $2,995.*

Reader Service No. 28



Analogic SSH/DSP DACs

High-speed DSP chip set

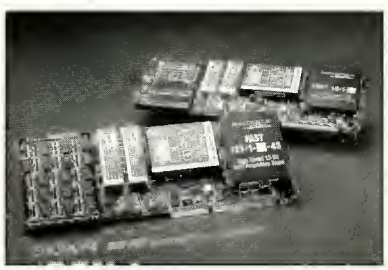
A 25-MHz, 24-bit fixed-point DSP, the LH9124LY processes 8- to 24-bit data in real time and performs digital filtering, image recognition, compression, spectrum analysis, correlation, convolution, and adaptive filtering in the time and frequency domains. Featuring six on-board multiplier/accumulators, eight adders, four complex bidirectional buses, and 24-bit external and 64-bit internal precision, the chip set handles very large arrays, 2D arrays, or data from up to 32 independent channels. Supported by the LH9320LU-25 address generator, the 26-function device performs a radix-16 butterfly in one pass and a 1K complex FFT in 129 µs. Built using a 0.8-µm, double-metal CMOS process and packaged in a 262-lead PGA, three cascaded LH9124LYs perform a complex 1K FFT in 41 µs. Typical applications include ultrasound, spectrum analysis, and tomography. *Sharp Electronics; $760 (100s, LH9124LY-25), $125 (100s, LH9320LU-25).*

Reader Service No. 29

Image acquisition

Vision-EZ, a Windows-based image acquisition package, uses a frame grabber and software to display live video and capture, save, and print images. Using 640×480 square pixel spatial resolution and 256 gray levels, Vision-EZ captures images in real time from video cameras, VCRs, and still-video devices. Accompanying Global Lab Acquire software lets users save images in TIFF, PCX, or DT-IRIS formats. With four on-board image-enhancing input LUTs, the board handles arithmetic, contrast adjustment, and reverse video functions. *Data Translation; $995.*

Reader Service No. 30



Data Translation's Vision-EZ

Analog input card

An 18-bit data acquisition system for the PC/104 bus, the 4A22 boasts 16 conversions per second, with an integration time that can be set to maximize 50- or 60-Hz rejection. Featuring software-selectable 0.5V or 5V full scale analog input ranges and 14 input channels with 500V isolation, the 4A22 stores calibration data in an on-card EEPROM, eliminating the need for zero- or full-scale adjustment potentiometers. For maximum accuracy, an on-card temperature readout channel allows for software compensation of reference voltage drift, while galvanic isolation of input signals eliminates common ground loop problems. Driver software for background data collection is included. *Mesa Electronics; $171 (100s).*

Reader Service No. 31

CAD tools

ASIC standard cell library

Using designs generated with the AnaCMOSLib V5.0 layout library, manufacturers can fabricate chips in 2- to 0.8-µm, double-metal CMOS-bulk technologies. This analog/neural network ASIC layout library upgrade supports AMI, ES2, Hewlett-Packard, Orbit, US2, and VLSI vendors, and can run on IBM-PC and compatible, Sparc, HP-9000, and Macintosh systems. New elements include transmission gate multiplexer; transmission gate with active high, with active low enable, and with complementary enables; poly resistor; and N- and P-channel FETs. *Tanner Research; from $495 (DOS systems).*

Reader Service No. 32

Thermal modeling for Pentium

Using CFD techniques to characterize thermal behavior, Flotherm software helps designers package Intel's Pentium, which dissipates up to 16 watts of heat, almost double its predecessors. Based on a 3D solution of equations governing airflow around components as well as conduction within, Flotherm models help define new system cooling requirements at an early development stage. *Flomerics; yearly licenses from $11,000.*

Reader Service No. 33

Integrated design environment

ASIC Powertools incorporates various design tools in one package for VHDL design, timing analysis, and test applications. An integrated multi-methodology design flow comprised

READER SERVICE CARD

IEEE MICRO

October 1993 issue (card void after April 1994)

Name _____

Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Country _____ Phone(_____) _____

Reader Interest
(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150	151	152	180	181	182
153	154	155	183	184	185
156	157	158	186	187	188
159	160	161	189	190	191
162	163	164	192	193	194
165	166	167	195	196	197
168	169	170	198	199	200
171	172	173	201	202	203
174	175	176	204	205	206
177	178	179	207	208	209

Product Information
(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	
11	31	51	71	91	111	131	
12	32	52	72	92	112	132	
13	33	53	73	93	113	133	
14	34	54	74	94	114	134	
15	35	55	75	95	115	135	
16	36	56	76	96	116	136	
17	37	57	77	97	117	137	
18	38	58	78	98	118	138	
19	39	59	79	99	119	139	
20	40	60	80	100	120	140	

READER SERVICE CARD

IEEE MICRO

October 1993 issue (card void after April 1994)

Name _____

Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Country _____ Phone(_____) _____

Reader Interest
(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150	151	152	180	181	182
153	154	155	183	184	185
156	157	158	186	187	188
159	160	161	189	190	191
162	163	164	192	193	194
165	166	167	195	196	197
168	169	170	198	199	200
171	172	173	201	202	203
174	175	176	204	205	206
177	178	179	207	208	209

Product Information
(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	
11	31	51	71	91	111	131	
12	32	52	72	92	112	132	
13	33	53	73	93	113	133	
14	34	54	74	94	114	134	
15	35	55	75	95	115	135	
16	36	56	76	96	116	136	
17	37	57	77	97	117	137	
18	38	58	78	98	118	138	
19	39	59	79	99	119	139	
20	40	60	80	100	120	140	

SUBSCRIPTION CARD

SUBSCRIBE TO IEEE MICRO

All the facts about today's chips and systems

❑ **YES**, sign me up!

If you are a member of the Computer Society or any other IEEE society, pay the member rate of only $23 for a year's subscription (six issues).

Society: _____

IEEE membership no: _____

Society members: Subscriptions are annualized. For orders submitted March through August, pay half the full-year rate ($11.50) for three bimonthly issues.

❑ **YES**, sign me up!

If you are a member of ACM, ACS, BCS, IEE (UK), IEEE but not a member of an IEEE society), IECEJ, IPSJ, NSPE, SCS, or other professional society, pay the sister-society rate of only $42 for a year's subscription (six issues).

Organization: _____ Membership no: _____

For information on airmail option, send fax request to (714) 821-4010.

_____
Full Signature Date

Name _____

Street _____

City _____

State/Country ZIP/Postal Code

❑ Payment enclosed *Residents of CA, DC, Canada, and Belgium add applicable sales tax.*

❑ Charge to ❑ Visa ❑ MasterCard ❑ American Express

Charge-card number

_____

Expiration date

____ ____
Month *Year*

Prices valid through 12/31/93

1093 MICRO

Charge orders also taken by phone:
(714) 821-8380 8 a.m. to 5 p.m. Pacific time
Circulation Dept.
10662 Los Vaqueros Circ., PO Box 3014
Los Alamitos, CA 90720-1264

Editorial comments

I liked: _____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

Reviewers Needed. *If interested, send professional data to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129, Torino, Italy.*

For reader service inquiries, see other side.

PLACE POSTAGE HERE

PO Box is for reader service cards only.

IEEE Micro
PO BOX 16508
NORTH HOLLYWOOD CA 91615-6508
USA

Editorial comments

I liked: _____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

Reviewers Needed. *If interested, send professional data to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129, Torino, Italy.*

For reader service inquiries, see other side.

PLACE POSTAGE HERE

PO Box is for reader service cards only.

IEEE Micro
PO BOX 16508
NORTH HOLLYWOOD CA 91615-6508
USA

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 38 LOS ALAMITOS, CA

POSTAGE WILL BE PAID BY ADDRESSEE

IEEE COMPUTER SOCIETY
PO BOX 3014
LOS ALAMITOS CA 90720-9804
USA

of graphics- and language-based design entry, synthesis, mulitlevel VHDL and structural simulations, timing analysis, test analysis, and ATPG, the suite of tools comes in two configurations: ASIC Architect and ASIC Expert. Tools include design composition, multilevel simulation, VHDL simulator, Motive integration for static timing analysis, and Sunrise test solution integration applications. *Viewlogic; from $70,000.*

Reader Service No. 34

ASIC system design tools

A library of 1.0- and 0.8-µm gate array and embedded cell array families now supports the Synopsys VHDL System Simulator. Offering debugging and analysis tools to isolate and resolve coding errors, VSS V3.0 lets designers capture concepts, verify high-level specifications, and detect design inconsistencies before committing to gate-level implementation. Designed using the Synopsys Library Compiler, the design kit boasts 12K to 400K raw gates at the 0.8-µm gate level, and 6K to 70K raw gates for 1.0-µm designs. The 0.8-µm devices clock at up to 100 MHz, dissipate power down to 2.4 µW/MHz/gate, and run at 215 ps for a two-NAND gate. Available in plastic, ceramic, or power QFPs and 576-pin TAB packages, all gate arrays and embedded cell arrays feature a four-alternative speed/power option, selectable slew rate, and both 3V and 5V fully characterized cell libraries.

Also available is a design kit to support Mentor Graphics V8.2 for top-down design of these 0.8- and 1.0-µm arrays. Supported are Top Down Design-Solver, QuickFault II, FLMS, a Falcon Framework-based application for library development and management, and X terminal support. *Mitsubishi Electronics America.*

Reader Service No. 35

Yield-enhancement software

To help resolve questions about design-process mismatches that reduce yields, Data Mapper combines design, test, inspection, and process data in a statistically rich graphical environment. Communicating with scanning electron microscopes and focused ion beam equipment running Framework, this approach characterizes and images defects to help determine their source in the IC manufacturing process. For each set of defined wafers and variables, Data Mapper generates a display consisting of wafer maps, histograms, defect images, and the IC design layouts of each physical die layer. The Bit Mapper option automatically relates defect data and images to bit failures. Data Mapper imports data from SQL databases, tester-specific formats, ASCII files, and CAD data formats. *Knights Technology.*

Reader Service No. 36

Knights Technology's Data Mapper

VHDL design simulator

Developed jointly by Mentor Graphics and Model Technology is the QuickVHDL Simulation Family. QuickVHDL and its Pro Systems and Pro IC options offer direct compiled code simulators for the ASIC, IC, and system design environments. Integrated with Design Architect software for high-level graphics entry and automatic VHDL generation, the family also works closely with AutoLogic for synthesis and supports the Std_DevelopersKit. Supplied with design extractor, synthesis compiler, and cosimulation interface features, it compiles VHDL into generic RISC instructions, which then get mapped into the instruction set of the workstation that is executing the simulation. Designed for Sparc and HP-PA platforms, QuickVHDL also offers a Pro System configuration for accessing AMP-based ASIC libraries and board-level models with QuickSimII. The Pro IC option runs M models, gate-level Lsim primitives, switch-level models, SPICE models, and switched capacitor models via the Lsim simulator. *Mentor Graphics; $19,950 (QuickVHDL), $29,950 (QuickVHDL Pro System, Pro IC).*

Reader Service No. 37

Expanded CBiCMOS cell library

With 35 new cells, the RSC4000 mixed-signal cell library gives designers of video processing, instrumentation, and communications applications an expanded set of mixed analog and digital application-specific tools. Consisting of analog bipolar, CMOS logic, and data conversion cells, the library uses a 2-µm CMOS and complementary bipolar process with 4-GHz NPN and 1.5-GHz PNP transistors and thin-film resistors. Additions include CMOS logic gates, flip-flops, multiplexers, and I/O devices, as well as bipolar ECL logic cells. Typical design specifications include a 2-nV/root Hz noise performance, 200-MHz signal processing bandwidths, and 2,000-V/µs slew rates. *Raytheon.*

Reader Service No. 38

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 189 Medium 190 High 191

Product Summary

Joe Hootman
University of North Dakota

Manufacturer	Model	Comments	R.S.#
Boards			
Bit 3 Computer	Bus-to-bus adapters	RISC/Unix family's common VMEbus card and support software driver permit application software portability from Sparcstations, RS/6000s, HP 9000s, DECstation 5000 Series 700s, and the Silicon Graphics Indigo to the VMEbus. *$2,850 each.*	80
Delta Computer Systems	MC 186/40 board	Improved four-axis motion control board offers fast, precise position control of hydraulic and electric servo systems that use the Multibus I or the Reliance AutoMate programmable controller. Coordinated position and speed changes can be made on the fly.	81
Microway	Quadputer-860 processor	Four i860 RISCs configured as a shared-memory MIMD device provide 200-Mflops numeric performance on EISA PCs. Five 25-MHz, 32-Mbyte Quadputers used in one EISA workstation provide an aggregate throughput of over 1 Gbyte (requires Unix). *$11,955.*	82
Systems			
Tektronix	GPX/DAS logic analyzer	Logic analyzer with full disassembly capability and probe adapter supports Intel Pentiums on the general-purpose GPX analyzer for medium-size design tasks and the DAS 9200 systems analysis platform for multiprocessor design. Both run at 66 MHz and offer real-time tracing, performance analysis, and 10-ns triggering.	83
Software			
Apple Computer	PlainTalk tool kit	Developer's tools create Macintosh-based applications that convert typed text to spoken English through concatenative Text-to-Speech technology, an extension of System 7. The tool kit contains Text-to-Speech Manager, various voices, and two engines: basic-level MacinTalk 2 and high-quality-voice MacinTalk Pro 2. A Macintosh Plus or greater is required. *$1,000 to $1,500 per year (worldwide redistribution license), depending on engine.*	84
Hippo Software	Hippix package	Set of commands and programming library allow Unix users to integrate OS/2 2.0 and Windows NT personal computers into their Unix networks. The command set implements most of IEEE Posix Std. 1003.2 and 1003.2a. The library supports over 90% of Posix 1003.1 API functions. *$239; $179 (commands only).*	85
Miscellaneous			
Telex Communications	MagnaByte M2X panel	Multimedia LCD projection system contains a built-in audio system with speaker and amplifier, Action! Windows/Macintosh presentation software, cables, and carrying case. PC users can plug in an interface bus to upgrade for Windows digital audio support. Users with Notebook and other portables can add audio to presentations without added hardware. *From $5,195 .*	86

Micro Review

continued from p. 5

cedures that you can find documentation for. One of the Visual Basic help files contains declarations for all of the DLL procedures in the Windows 3.1 application programming interface. You can simply press a button to copy them from the help file and paste them into your Basic program.

Visual Basic also lets you use the Windows object linking and embedding (OLE) feature to incorporate items like spreadsheets or graphics in your application. You can manipulate them from within your application or see the updated versions when they change in their original locations.

Once you get hooked on Windows programming with Visual Basic, you can play to your heart's delight with its more advanced features. Some are as mundane as interfacing to a relational database using SQL. Others, like three-dimensional buttons or animated icons can exercise your creativity.

If you want to be able to throw together an ad hoc Windows application the way you used to do with Basic on your CPM system, this is the package for you.

Running Visual Basic 3.0 for Windows, Ross Nelson (Microsoft Press, Redmond Wash., 1993, 346 pp.; $22.95)
Microsoft Visual Basic Workshop, Windows Edition, John Clark Craig (Microsoft Press, Redmond Wash., 1993, 504 pp. plus diskette; $39.95)

These two books take different approaches to introducing you to Visual Basic. Nelson takes you systematically through the features, concocting ad hoc examples appropriate to each topic. Craig builds realistic applications, each emphasizing a particular feature, then explains each in detail. If you're serious, get them both. If not, get Nelson's book; it's shorter and cheaper.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 183 Medium 184 High 185

Classified Advertisement

continued from p. 3

models of legal protection. (Such recent decisions include those in the *Galoob, Sega,* and *Altai* cases.) These recent decisions have also reemphasized the principle of copyright law that predominantly functional and utilitarian aspects of works are not protected by copyright.

Model difficulties

The difficulties with the two principal models, patents and copyrights, center on the fact that they do too little—and do too much—for software and its noncode aspects. For example, in principle neither copyrights nor patents protect ideas. Yet, some of the most creative and valuable aspects of software may be classified as ideas. At least they are defined at so high a level of abstraction that they are, for purposes of patent law, unprotected ideas or mathematical principles. For purposes of copyright law, they are unprotected ideas, procedures, processes, systems, methods of operation, concepts, principles, or discoveries. Algorithms and user interfaces are salient examples. So, too, are languages and instruction sets. The patent and copyright models, unless they are greatly altered, do not allow for protection of highly abstract subject matter.

The fairly rigid requirement of the two traditional intellectual property legal models that injunctions must be issued against infringement creates problems, too, for the idea aspects of software. Congress and the courts have been faced with the choice of either ordering a total prohibition against any use by the defendant of the plaintiff's allegedly protected software idea or else making a determination that the allegedly infringing material is outside the scope of the plaintiff's statutory right. Both bodies have often preferred the underprotection of the second alternative to the overprotection of the first. Because the copyright and patent models insist on doing too much for copyright and patent owners, in some ways, our present laws may give too little to the creators of new software or those who invest in bringing it to the market. On the other hand, the alternative under the two existing models would be, presumably, worse because it is excessive.

Indeed, the principal models of intellectual property protection not only at times do too little or do too much for the idea aspects of software, but they are often completely askew to software. Software and the intellectual property laws are often like ships passing in the night. Or perhaps you will prefer the metaphor of the Second and Ninth Circuits in the *Altai* and *Sega* cases: Using one of them for software is like "forcing a square peg into a round hole."

There are other legal models for industrial property protection, however. Copyright and patent law do not exhaust the known legal schemes for protection of intangible rights in intellectual creations.

Another model

Other legal models are known in other countries. One alternative to consider for industrial property protection of software is that of the utility model, known in many European countries and Japan. A utility model is a kind of petty patent. It typically has narrower scope, it may be allowable upon mere registration with a government agency, its validity is sustained on the basis of a lesser showing of technical merit than a regular patent, and it may have a shorter life span. The US Semiconductor Chip Protection Act of 1984 is basically a utility model law, and represents the first major departure in the US from the patent and copyright models of intellectual property law. Some of this law's provisions may suggest a more appropriate legal scheme for protection of computer software than the patent and copyright models do.

Using utility model laws as an alternative model for industrial property protection suggests that some of the following features for a software protection law directed to noncode aspects of computer software and nonverbatim copying of computer programs may be appropriate. The following sketch of an industrial property law for protecting rights in future software technology presents a tentative model. I often use the word *probably.* Where I do not, you should understand it as implicit. This is a matter that calls for studying, discussing, and identifying the interests at stake, balancing them, and making appropriate trade-offs and a considerable number of other compromises. At this stage, it is more appropriate to raise questions, identify issues, and point to a possible model than it is to purport to prescribe definitive solutions.

First, something like utility model protection could be afforded to anything in the *index prohibitorum* of section 102(b) of the US Copyright Act. Any ideas, procedures, processes, systems, methods of operation, concepts, principles, or discoveries are excluded by copyright law from copyright protection. But Congress is free to protect them under a different law if that appears wise. By the same token, patent law's exclusion of laws of nature and mathematical principles (for example, algorithms) need not be a limitation on property rights in a utility model. Of course, common sense and considerations of policy may call for some of the same limits, but that would be a matter addressed to the wisdom of the legislature, not its power.

I assume utility model protection would be considered for algorithms, languages, instruction sets, icons, and some aspects of dataflow schemes and so-called sequence, structure, and organization of computer programs. More generally, utility model protection could be provided for any economically valuable aspects of future software technology for which Congress judged that industrial property protection is needed to encourage their creation, disclosure, and commercialization.

Protection of such aspects of computer software would probably be incompatible with adoption for software of the full scope of patent and copyright relief and remedies now in use. For example, injunctions against use of such software features by others may well be inconsistent with promotion of rapid software progress. Moreover, locking up new algorithms or instruction sets tightly for a period of years is probably a mistake.

It is one thing to make users pay reasonable compensation for the use of such new ideas, but quite another thing to permit their creators to withhold their use at will or levy any charge they please, with the aid of state compulsion. That means that a scheme for just and reasonable compensation—similar to that applied in the Federal Claims Court when the government uses or takes intellectual property rights belonging to others—would need to be considered.

As in the case of the semiconductor chip law and many utility model laws, an intermediate level of required technical merit would probably be appropriate. That would be one between patent law's inventive step and copyright law's *de minimis* standard. However, in-depth evaluation of the technical merit of particular software probably should be saved for litigation, if it ever occurs, as our semiconductor chip law provides.

A utility model law for software would probably call for registration of software rights, and immediate attachment of industrial property rights, while avoiding the high front-end costs of a patent-type examination procedure. The expense of a thorough analysis of the prior work in the field and of the advance over prior work embodied in the software feature in question would therefore be deferred until a plaintiff and a defendant are concerned enough about the matter to be willing to spend the kind of money involved in a lawsuit.

(There are possible arguments on the other side, however, as well. It may be thought that the potential terrorism effect—in legal parlance, *in terrorem* effect—of intellectual property rights calls for advance screening of validity issues, so that would-be competitors are not scared off. I think, while others do not, that the social cost of legal terrorism is substantially exceeded by the social cost of having to make a thorough validity study of the 99.9 percent of software innovations that will never become commercially important and stir up litigation.)

The scope of software rights should probably be more like those of patent law than copyright law. Control over use is a major right omitted by copyright law, doubtless for good reasons in the case of copyright's traditional subject matter, but the right is economically important for software. Hence, software rights, like patent rights, should include control of use of the protected subject matter. Such uses would include, for example, execution of programs implementing protected algorithms or written or protected programming languages.

Certainty as to scope of industrial property rights is desirable. But the specificity of patent law's claims may be too difficult and expensive to emulate. On the other hand, the completely open-ended nature of copyright—what you see is what you get, and a court will eventually decide what that is—is also problematical. Some sort of intermediate compromise is in order, but I am unable to describe it for you. This aspect of the system needs more study. (So do most of the other features of the proposed model. This is a proposal for thinking about the issues, not an off-the-shelf solution.)

Independent creation of algorithms and other software features is another issue in search of a good solution. Like scope, it calls for deliberation and perhaps compromise. Patent and copyright laws have opposite answers to offer on this point. (As indicated earlier, independent creation of the infringing subject matter is an absolute defense to a claim of copyright infringement, and is no defense at all to a claim of patent infringement.) The US chip law was, unfortunately, silent on the issue.

Software rights, like patent rights, should include control of use of the protected subject matter.

Clearly, software utility model law will not draft itself. It is a substantial undertaking. But the expenditure of effort in undertaking study and drafting is worthwhile. It is a better course than trying to sweep the difficult issues under the rug by purporting to incorporate by reference some other existing body of law in its entirety. That is what the proponents of the 1984 Semiconductor Chip Protection Act initially tried to do, and in fact persuaded the Senate to endorse. (The Senate passed a chip bill that simply extended copyright law to cover chip layouts.) But the chair of the Intellectual Property Subcommittee in the House of Representatives (Robert Kastenmeier) wisely blocked that move. His opposition to a copyright incorporation-by-reference solution forced a choice of a so-called *sui generis* chip law. (*A sui generis* intellectual property law is one that does not fit under either traditional copyright law or tranditional patent law. It is like copyright law in some respects, like patent law in some respects, and like neither of them in other respects.)

The *sui generis* chip law had to be specially drafted to address the specific perceived problems of the semiconductor industry. Unfortunately, drafting a sound software protection law is probably one or two orders of

magnitude more difficult than drafting a chip protection law.

Modifying existing laws

In any discussion of adopting other legal models for software, one may ask several questions. Why depart from the copyright and/or patent law models at all? Would it not be preferable just to modify one or both of these existing schemes to put some corners in the holes so that they will accommodate square pegs? Are they not familiar legal schemes with established, known bodies of case law helping us to fill in necessary omissions or ambiguities in any statute applied to software? And why dispense with the international recognition of intellectual property rights classified as copyrights or patents, and thus forego automatic transnational protection of software rights under international treaties?

Congress answered many of these questions in the *House Report on the Semiconductor Chip Protection Act of 1984*. The House refused to follow the Senate's copyright approach to chip protection and insisted on enactment of a *sui generis* chip law instead (H. Rep. No. 98-781, 98th Cong., 2d Sess. 7-8, 1984). It recognized that, to a significant extent, the Berne Convention and the existing body of US copyright and patent law do not permit us to square off the legal hole to make the software peg fit it. (For example, the Berne Convention does not permit reduction of the term of copyright protection substantially below its present 50- to 75-year term. Nor does it permit a copyright law to "discriminate" against software, literary-work copyrights by denying them injunctions and, in effect, opening them up to the compulsion of unconsented-to use. Protection of abstract ideas and systems is infeasible under our existing intellectual property laws.)

Indeed, to modify copyright law and patent law sufficiently so that they accommodate software properly would mean changing them so much that they would no longer be copyright or patent law. Not only would that be infeasible, but it would be unacceptable to the users and beneficiaries of traditional copyright and patent law.

International protection is speculative, apart from verbatim copying. There is no automatic transnational copyright protection, apart from verbatim or near-verbatim copying of programs. There is no worldwide consensus in favor of protecting nonliteral aspects of software under copyright law. Indeed, in the wake of the *Altai* and *Sega* decisions, there may no longer be such domestic protection.

Moreover, if the US tried to protect ideas and other abstract or nonliteral aspects of software under one or both of the two principal existing systems, patents and copyrights, serious trade problems would occur. US treaty obligations would then oblige us to protect such aspects of foreign nationals' software while their home countries continued to refuse to protect the same aspects of US companies' software. That is required by the principle of equal "national treatment," the cornerstone of our intellectual property treaties. (See Berne Convention, art. 5(1); Paris Convention, art. 2(1).) As one scholar has pointed out: "A unilateral broadening of protection at home can thus weaken a [treaty] Member State's overall competitive position."

Finally, the concept of saving effort by relying on the existing body of copyright law is an illusion. Earlier, I gave a few examples of difficult choices that may need to be made, in sketching the outline of a software law based on the legal model of a utility model. There are many more difficult choices. They should be addressed explicitly, not by pretending that some existing body of law that has never addressed, or had any reason to address, such issues offers a ready-made solution to be incorporated by reference. The likelihood of coming to rational solutions for these problems is more enhanced by addressing them explicitly, than by hoping that buying a lottery ticket—meaning incorporating a preexisting law by reference—will lead to a serendipitous outcome. (Murphy's Law is inconsistent with your getting free lunches.) The right way to craft an appropriate scheme of industrial property protection for software is to do it purposefully.

What's next?

The most recent trend of decision in software copyright cases is away from *Whelan*, away from protection of nonliteral aspects of computer programs, and away from treating copyrights as if they were patents. That does not mean that industrial property protection of nonliteral, relatively abstract aspects of software is a bad idea. Nor does it mean that courts should or do consider these aspects of software to be without economic value and undeserving of any kind of industrial property protection. It means only that the traditional intellectual property law models, particularly that of copyright law, do not accommodate such protection. We need to consider another model. That in turn would mean that legislation of some kind is needed to accord such protection.

Before legislation is attempted, however, much more study and thought are needed. Legislation should only follow both study and an attempt to build an informed consensus among software professionals. That implies developing a consensus in crafting, and then in support of, an appropriate law for industrial property protection of the valuable aspects of software technology. How could or should that be done?

Congress attempted to address this question in the 1970s. It established a National Commission on New Technological Uses of Copyrighted Works (CONTU) to study and advise Congress what to do about protecting computer software. CONTU presented its final report to Congress in 1979. CONTU's majority (over a strong dissent) punted in favor of letting the status quo develop further, under the copyright law, and Congress accepted that recommen-

dation. That resulted in the last decade or so of total confusion and legal mayhem in the software protection field.

Efforts since then within the IEEE Computer Society's Committee on Public Policy (COPP) and IEEE-USA's Intellectual Property Committee have not been notably successful. Perhaps, the IEEE is not institutionally well suited for this kind of process. The IEEE seems to work better with reactive projects than proactive projects.

Possibly, academia is more well suited to nuture the proposed computer software law project. But then there is the issue of overcoming distance (or at least perceived distance) from reality. My present view is that a university setting is more suited to this process (at least initially) than any other. But academia needs a very large amount of nonacademic leavening to accomplish satisfactory, realistic results.

By the time this issue of *IEEE Micro* reaches its readers, one preliminary effort in this general direction will have occurred. On October 1, 1993, the George Washington University's School of Engineering and Applied Science (GWU SEAS), in cooperation with the University of Wisconsin's Kastenmeier Foundation, is hosting a one-day Technology Workshop on Computer Software Protection. The workshop chair is Gideon Frieder, dean of GWU SEAS, who has long been interested and involved in computer software/intellectual property law problems.

I have assisted the dean in this project for several reasons. One is that I am the person who teaches computer software copyright and patent law at GWU. Moreover, I have been concerned professionally about these issues for 25 years. I have become more and more concerned—and indeed apprehensive—that the US may be going in the wrong direction in this field, with a likely adverse consequence, among others, of loss of its world leadership of computer software progress. The time has come to subsitute positive action for mere concern.

The workshop participants are a mixture of academicians (mainly EE and CS Department), computer industry business representatives, other computer science professionals, and government representatives, along with a minimal sprinkling of intellectual property lawyers. The plan is to describe and discuss computer software technology of the future (1990s-2010). At the outset, we need to ascertain whether and to what extent, if any, computer software evolution will strain present intellectual property law; consider possible effects on software progress and the public interest; and consider whether another legal model for computer software protection is needed, or at least needs to be considered.

The purpose of the workshop is more to ventilate views and open the door to future discussions and study than to reach firm conclusions. Some workshop participants are known to favor the legal status quo, and to feel that we only need to give present copyright law (and patent law) more time and opportunity to evolve and develop. These participants favor evolution of computer software protection by the judicial route—let the courts work out proper software protection, on a case-by-case basis.

Others have different views; readers of this column will know that I am among those skeptical about the likelihood of success of that course of action, and believe in legislation by Congress rather than by the courts. The organizers of this workshop consider it more important simply to discuss the issues and explore the range of opinions than to determine which views are right and which wrong—if that is even a meaningful notion in this context.

Further workshops are planned to follow up on various issues, possibly over a span of several years. As the earlier part of this column suggests, it is unclear what the best answers are to many questions about the model and architecture. We hope a more informed view will result from further study and discussion, and that a clearer description of the appropriate model for legally protecting computer software will emerge. Ideally, the train of discourse will end with one of two proposals. We could propose a federal computer software protection law for algorithms, instruction sets, programming languages, and other nonliteral aspects of computer software technology, based on an improved legal model. Or, we could abandon the whole project as quixotic or infeasible. Perhaps there are other possibilities.

It is more important simply to discuss the issues and explore the range of opinions than to determine which views are right and which wrong.

Readers who would like to participate actively in this project in 1994 can write Gideon Frieder, School of Engineering and Applied Science, George Washington University, 725 23rd St. NW, Washington, DC 20052; frieder@seas.gwu.edu. You can also write to me at the address shown on the first page of this column.

FOR DISPLAY ADVERTISING INFORMATION, CONTACT:

Southern California and Mountain States: Richard C. Faust, Douglas C. Faust, 24050 Madison Street, Suite 101, Torrance, California 90505; Tel: (310) 373-9604; Fax: (310) 373-8760.

Northern California and Pacific NW: William W. Hague, 9017 Peacock Hill Road, Gig Harbor, Washington 98332: Tel: (206) 858-7575; Fax (206) 858-7576.

East Coast: Gail A. Frank, Nancy Inserra, 82 Bethany Road, Suite 4, Hazlet, New Jersey 07730; Tel: (908) 264-1100; Fax: (908) 264-4340.

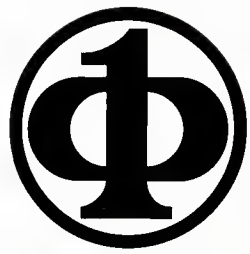
New England: Paul Gillespie, PO Box 6444, Holliston, Massachusetts 01746; Tel: (508) 429-8907; Fax: (508) 429-8684.

Southwest: Frank E. Johnson, 3601 Smith-Barry Road, Suite 103, Arlington, Texas 76013; Tel: (817) 275-2651; Metro Voice/Fax: (817) 265-3811.

Midwest: Harold L. Leddy, 345 Auburn Avenue, Winnetka, Illinois 60093-3603; Tel: (708) 446-8764; Fax: (708) 446-7985.

Advertising Manager: Heidi Rex, 10662 Los Vaqueros Circle, Los Alamitos, California 90720-1264; Tel: (714) 821-8380; Fax: (714) 821-4010.

For production information, conference, and classified advertising, contact Marian Tibayan. *IEEE MICRO*, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, California 90720-1264; Tel: (714) 821-8380; Fax: (714) 821-4010; email: m.tibayan@compmail.com.



IEEE COMPUTER SOCIETY®

A member society of the
Institute of Electrical and Electronics Engineers, Inc.

THE FOLLOWING INFORMATION IS AVAILABLE:

Contact the Publications Office;
to facilitate handling, please request by number.

- Membership application, student #203, others #202
- Publications catalog #201
- Technical committee list/application #197
- Chapters lists, start-up procedures #193
- Student scholarship information #192
- Volunteer leaders/staff directory #196
- IEEE senior member grade application #204
  (requires ten years practice and significant performance in five of those ten)

To check membership status or report a change of address, call the IEEE toll-free number, 1-800-678-4333. Direct all other Computer Society-related questions to the Publications Office.

PURPOSE

The IEEE Computer Society advances the theory and practice of computer science and engineering, promotes the exchange of technical information among 100,000 members worldwide, and provides a wide range of services to members and nonmembers.

MEMBERSHIP

Members receive the acclaimed monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

PUBLICATIONS AND ACTIVITIES

Computer. An authoritative, easy-to-read magazine containing tutorial and in-depth articles on topics across the computer field, plus news, conferences, calendar, interviews, and product reviews.

Periodicals. The society publishes eight magazines and seven research transactions. Refer to membership application or request information as noted at left.

Conference Proceedings, Tutorial Texts, Standards Documents. The Computer Society Press publishes more than 100 titles every year.

Standards Working Groups. More than 100 of these groups produce IEEE standards used throughout the industrial world.

Technical Committees. More than 30 TCs publish newsletters, provide interaction with peers in specialty areas, and directly influence standards, conferences, and education.

Conferences/Education. The society holds about 100 conferences each year and sponsors many educational activities, including computing science accreditation.

Chapters. Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

OMBUDSMAN

Members experiencing problems — magazine delivery, membership status, or unresolved complaints — may write to the ombudsman at the Publications Office.